НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Система захисту даних на базі алгоритму Ель-Гамаля

Виконав: студент 4 курсу, групи ТВ-51

Савчин Юхим Васильович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доц. Крячок Олександр Степанович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.т.н., доцент Реуцький М.О. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Савчину Юхиму Васильовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “ Система захисту даних на базі алгоритму Ель-Гамаля”

керівник роботи к.т.н. доц., Крячок Олександр Степанович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи бінарний файл з розширенням .wt, в якому зберігаються результати шифрування даних.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати математичну модель методу Ель-Гамаля, реалізувати модуль шифрування, та реалізувати web-сервіс шифрування-дешифрування текстових даних та зберігання оброблених даних у базі даних MongoDB.

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень) 1. Мета та завдання роботи 2. Огляд існуючих рішень 3. Функції керуючого модуля шифрування-дешифрування 4. Функції Web-сервісу «Information security system» 5. Формули розрахунку 6. Використані програмні засоби 7. Висновки

Дата видачі завдання "___"_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Савчин Ю. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Крячок О. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи було створення системи шифрування–дешифрування текстових даних та їх безпечне зберігання у базі даних. Система генерує ключі, необхідні для використання методу Ель–Гамаля, та присвоює їх індивідуально кожному користувачеві системи. Керуючий модуль системи забезпечує можливість використання методу шифрування–дешифрування даних та взаємодії клієнтського додатку з базою даних, яка зберігає ключі, та користувацьку інформацію у зашифрованому вигляді. Користувацький додаток представляє собою систему електронної лікарні з приватним кабінетом у якому пацієнт може у будь–який момент отримати інформацію про його захворювання, діагнози та отримані від доктора рекомендації про лікування.

Записка містить 74 сторінки 22 рисунка 4 формули та 28 посилань.

ABSTRACT

The purpose of the work was to create a system of encryption–decryption of text data and their safe storage in the database. The system generates the keys needed to use the El Gamal method, and assigns them individually to each user of the system. The system's control module provides the ability to use the encryption–decryption method and the interaction of the client application with the database that stores the keys, and the user information in encrypted form. The custom application is an electronic hospital system with a private office in which the patient can at any time receive information about his illness, diagnoses and advice received from the doctor about treatment.

The note contains 74 pages 22 images 34 formulas and 25 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕЦП – електроний цифровий підпис;

СКБД – система керування базами даних;

API (англ. Application Programming Interface) – прикладний програмний інтерфейс;

БД – база даних;

CRUD (англ. create read update delete) – 4 базові функції управління даними «створення, зчитування, зміна і видалення»;

MVC – (англ. model–view–controller) – схема поділу програмного продукту, призначеного для користувацького інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер;

SPA - Single–Page Applications

ВСТУП

Стрімкий розвиток сучасного інформаційного середовища потребує ще більш ретельного вирішення задачі шифрування та дешифрування даних та їх використання неавторизованими користувачами.

Сучасна криптографія лежить в основі комп'ютерної безпеки та базується на різноманітних концепціях математики та інформатики, серед яких необхідно виділити теорію чисел, теорію складності математичних обчислень та теорію ймовірностей.

На сьогодні криптографічні методи використовуються у всіх системах, які працюють з цінними даними, а саме – у шифруванні транзакцій, банківській системі, допомагають зробити безпечною роботу пластикових карт, банкоматів та безпровідну передачу даних.

Основним компонентом криптографії є шифрування, яке використовує складні математичні алгоритми для перетворення вхідних даних в шифротекст у режимі шифрування, та шифротекст у вихідні дані в режимі дешифрування.

Пропонується розглянути програмну систему, яка дозволяє виконувати шифрування–дешифрування даних та їх зберігання на окремих серверах для більшої надійності системи.

Програма представляє собою Web–додаток, який надає сервіс зберігання текстових даних у зашифрованому вигляді.

Для надійності вирішено зберігати ключі, унікальні для кожного користувача у другій базі даних, що підвищує надійність системи.

Для шифрування даних пропонується використовувати алгоритм Ель–Гамаля, який забезпечує обмежений контроль доступа до даних що зберігаються у зашифрованому вигляді, та робить їх доступними тільки для користувачів, що мають секретні ключі. Також алгоритм Ель–Гамаля має ймовірнісні ключі, а тому і відносно велику стійкість в порівнянні із методами з незмінним процесом шифрування.

Записка містить 4 розділи.

У першому розділі проводиться огляд існуючих рішень шифрування–дешифрування текстових даних.

У другому розділі описані засоби реалізації програмної системи.

У третьому розділі дано опис програмної реалізації системи захисту даних.

У четвертому розділі описана методика роботи користувача з програмною системою.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ШИФРУВАННЯ–ДЕШИФРУВАННЯ ТЕКСТОВИХ ДАНИХ

Вся інформація, що зберігається в пам’яті вашого комп’ютера або в базі даних, може бути поділена на дві категорії:

— програмне забезпечення;

— дані.

Програмне забезпечення і додатки – це інструменти для роботи і обробки даних, а дані – це те, з чим працюють комп’ютерні програми. Звичайно, дані – найважливіше та найцінніше. Програми необхідні, але в дійсності цінність мають алгоритми і концепції, які вони використовують, а не самі додатки.

Більшість користувачів та експертів в області інформаційних технологій погоджуються: дані – це найважливіша складова роботи будь–якої програми і системи в цілому. Не важливо, урядові установи, банківські системи, телекомунікації, криптовалюта, медичні організації або транспортні служби, використання електронних даних – життєво важлива основа світової економіки.

Дані можна розділити на дві основні категорії:

— внутрішньо–спрямовані;

— зовнішньо–спрямовані.

Внутрішньо–спрямовані – це дані, які використовуються тільки всередені компанії чи сервісу, який вона надає. Ці дані необхідні для виробництва товарів та послуг. Внутрішні дані в більшості використовуються працівниками статистичних та інших ділових підрозділів компанії, тобто ця інформація не повинна виходити за межі інформаційної системи компанії.

Зовнішньо–спрямовані дані – це інформація, яка використовується для конкурування компанії чи сервісу на ринку. Ці дані використовуються при обслуговуванні клієнта та в аналітичній функції системи. Саме такі дані більш схильні до риску ніж внутрішньо–спрямовані [1, 2].

Захист даних та безпосередньо самі дані лежать в основі успішності компанії чи наданого сервісу та на прибуток, тому необхідно якісно та ретельно

підійти до вирішення проблеми шифрування–дешифрування всієї інформації, чи то внутрішньо, чи зовнішньо спрямовані дані.

На сьогодні в основі усіх технологій захисту даних лежать різноманітні комбінації математичних методів та інформатики. В цілому, шифрування – це оборотний процес перетворення інформації з ціллю її приховування від сторонніх лиць.

Для того, щоб інформація після обробки була схожою на нерозбірливий набір символів було розроблено спеціальні алгоритми, які поділяють на дві групи:

— симетричні;

— асиметричні.

До симетричних методів відносять такі алгоритми як AES, CAST, Blowfish, DES. Всі ці математичні рішення для шифрування та дешифрування використовують один ключ. Головний недолік таких методів – у разі викрадення ключа шифрування злодій може викрасти та повернути дані у початковий вигляд. Окрім цього є не мала кількість технологій криптоатак, які дозволяють дешифрувати дані навіть без використання ключів шифрування.

Асиметричні методи, такі як ElGamal та RSA, використовують пару ключів – приватний та публічний. Публічний ключ можна передавати по не захищеному каналу, та він призначений для перевірки ЕЦП та шифрування повідомлення. Для генерування ЕЦП та дешифрування повідомлення використовується приватний ключ. Такий підхід частково вирішує проблему перехвату ключів, адже навіть якщо зловмисник перехвате публічний ключ, це не дасть йому можливості дешифрування даних.

Варто зазначити, що кожна система може бути зламана простим методом перебору ключів, але у випадку з асиметричними методами зробити це буде значно важче як у плані програмної реалізації, так і у питанні покупки якісного та потужного серверного обладнання. Інший спосіб злому криптосистем – організація перехвату та аналізу повідомлень. Можливість криптосистем протистояти зламуванню називають крипостійкістю алгоритму шифрування,

тому при виборі методу для застосування у вашій системі варто приділяти увагу саме цьому показнику.

Варто зазначити, що саме асинхронні методи мають більшу крипостійкість у порівнянні з синхронними методами.

У сучасній криптографії надають перевагу методам, перевіреним часом, тому до сих пір для захисту даних активно використовують алгоритми RSA та Ель–Гамаля, які були розроблені у 1977 та 1985 роках відповідно [3-5].

Постійно розвиваються технології, а збільшення обчислювальних потужностей відкривають все нові і нові можливості для зловмисників. Деякі алгоритми, ще недавно вважалися стійкими, стають уразливі до атак простого перебору для пошуку ключа. Крім того, з появою нових технологій, наприклад хмарних обчислень, з’явилися абсолютно нові проблеми, що вимагають принципово інших підходів до їх вирішення. Криптографія, яка раніше вважалася засекреченою, на сьогоднішній день, стала вельми доступною. Адже кожен з нас неодноразово стикається з поняттями «ключ», «шифр», «криптограма». У цьому розділі пропонується розглянути основні підходи до шифрування та дешифрування текстових даних, та розглянути основні види алгоритмів, що застосовуються у криптографії сьогодні, або були популярними декілька десятків років назад.

1.1 Симетричне шифрування

Симетричні методи шифрування – найпростіші алгоритми для захисту даних, які як правило працюють лише з одним ключом як в режимі шифрування так і дешифрування. Цей ключ називають секретним ключем криптографії, (SKC), або спільним ключем. Симетричне шифрування має на увазі те, що секретний цифровий ключ повинен бути відомий як відправнику, так і отримувачу [3-5].

1.2 Асиметричне шифрування

Такі алгоритми широко використовуються як у локальних системах захисту даних, так і у Всесвітній мережі. Такі алгоритми оперують двома видами ключів – приватними та публічними та мають такі особливості:

— публічний ключ може бути відомим багатьом користувачам, але розшифрувати дані за його допомогою неможливо. Якщо проводити аналогію зі звичними речами, то публічний ключ являється аналогом електронної адреси;

— приватний ключ – секретний, який використовують для дешифрування інформації та ніколи не розкривають на іншій стороні. Наприклад, закритим ключом можна назвати пароль від облікового записку електронної пошти;

— немає значення, який ключ буде використовуватися першим, але для коректної роботи алгоритму в обидві сторони потрібні обидва ключа;

— дані можуть бути зашифрованими як за допомогою відкритого, так і закритого ключа.

1.3 Хеш–функції

Хеш–функції є алгоритмами, які в деякому сенсі не використовують ключ. Їх також називають дайджестами повідомлень або одностороннім шифруванням.

Такі алгоритми у більшості використовуються для перетворення великих об’ємів даних у шифротекст, а сам шифротекст виглядає як строка двійкових чисел (бітів), певної довжини (хеш), яку дуже важно імітувати. Таким чином хеш–функції забезпечують вимір цілісності надісланих файлів. Два різних повідомлення, що містять різну інформацію, не можуть мати однаковий хеш.

Хеш може використовуватися в якості цифрового підпису або для шифрування і зберігання паролів. Метод хешування є ключовим моментом технології блокчейн. Він в основному стосується захисту цілісності даних, що проходять через blockchain–мережі.

1.4 Блочний шифр

Блочний шифр є різновидом симетричного шифрування. Блочне шифрування має на увазі, що кожен блок даних шифрується або розшифровується окремо, причому кожен біт в вихідному блоці залежить від кожного біта в відповідному вхідному блоці, але не від інших бітів. Розмір блоку визначається алгоритмом. У більшості випадків блоки зазвичай мають 64–або 128–розрядний формат. Це означає, що їх розмір визначений і залишається завжди незмінним [6, 7].

1.5 Потоковий шифр

Потокові методи шифрування використовують симетричне шифрування. На відміну від блоку, де шифрування відбувається одночасно, потокове виконується по одному біту за раз. Перетворення символів відкритого повідомлення в символи шифрованого відбувається в залежності від їх розташування в потоці відкритого тексту і використовуваного ключа [8].

Потокове шифрування працює на дуже високій швидкості, а також має більш низьку апаратну складність. Важливим аспектом при використанні потокових шифрів є те, що вектор ініціалізації ніколи не повинен бути однаковим при відправці даних по мережі.

1.6 Цифровий підпис

Цифрові підписи найчастіше використовують асиметричну криптографію з відкритим ключем. Мають цифровий ідентифікатор на основі сертифікату, виданого акредитованим центром сертифікації (ЦС). Є частиною механізму перевірки безпеки, достовірності цифрових повідомлень [9].

Електронний документ, що містить ЦП, підтверджує аутентифікацію заявленого відправника, відповідає за цілісність переданих даних.

Електронні підписи зазвичай використовуються для проведення фінансових операцій, поширення програмного забезпечення, формування

податкових і бюджетних звітів, а також для виявлення підроблених документів або фальсифікацій. В основному для електронного підпису використовують асиметричні методи, такі як RSA та ElGamal.

1.7 Опис математичної моделі схеми Ель–Гамаля

Математична модель методу Ель–Гамаля та алгоритм роботи програми полягають у наступному [3–5]:

1. Генерується просте число $\boldsymbol{\rho}$, а також число $\boldsymbol{g}$ – первісний корінь числа $\boldsymbol{\rho}$.

2. Вибирається закритий ключ $\boldsymbol{x}$, такий, що $\boldsymbol{1 < x < \rho - 1}$.

3. Розраховується відкритий ключ $\boldsymbol{y = g^x \bmod \rho}$. (2.1)

4. Цих значень достатньо для того щоб почати шифрування, тому вибираємо випадковий сесійний ключ $\boldsymbol{k}$, такий що $\boldsymbol{1 < k < \rho - 1}$.

5. Повідомлення позначається буквою $\boldsymbol{\mathcal{M}}$. Розраховуємо числа:

$$\boldsymbol{a = g^k \bmod \rho} \quad (2.2)$$

$$\boldsymbol{b = y^k \bmod \rho}. \quad (2.3)$$

Ця пара чисел і є шифротекстом.

Для більшої захищеності файлів та тексту передбачено зберігання відкритих, закритих ключів та шифротексту у різних сховищах даних. Перетворення зашифрованого тексту до початкового виду виконується за допомогою найбільш використовуваної формули [3-6]:

$$\boldsymbol{\mathcal{M} = b(a^x)^{-1} = ba^{(\rho-1-x)} (\bmod \rho)}\ . \quad (2.4)$$

Оскільки при шифруванні використовуються випадкові величини, то метод можна назвати ймовірнісним шифром і тому він має велику стійкість у порівнянні з методами з незмінним процесом шифрування – з симетричними, блочними та потоковими методами шифрування.

Цей метод дозволяє обмінюватися даними між нескінченною кількістю абонентів, якщо вони мають спільний відкритий і особисті приватні ключі, але основною проблемою шифрування методом Ель–Гамаля є те, що шифротекст

має довжину у два рази більшу ніж початкова строка, а також те, що для більшої надійності треба використовувати прості числа довжиною більше ніж 128 біт, а по сучасним міркам рекомендовано використовувати 512 та 1024 бітні прості числа та числа RSA [10-12].

Висновки до розділу

В даному розділі розглянуто основні типи методів шифрування та описані відмінності між ними. Окрім цього описано метод Ель–Гамаля, який використовується в розроблюваному сервісі. Описані в розділі формули та підходи застосовано у Web–застосунку «Information security system».

2. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Аналізуючи поставлену задачу та методи її вирішення, вирішено розроблювати програмний комплекс на основі веб–технологій. Головною перевагою веб–застосунку перед іншими варіантами є його універсальність і можливість використання на будь–яких пристроях без портування на цільову операційну систему (браузер і його віртуальна машина виступає як цільова універсальна операційна).

Як основне середовище розробки обрано JetBrains WebStorm.

Для розробки серверної частини використовувалась мова програмування Node.js та фреймворк Express.

Для створення інтерфейсу користувача – фреймворк Nuxt для бібліотеки Vue.js.

У якості бази даних використано Mongodb

2.1 Середовище JetBrains WebStorm

Середовище розробки WebStorm з самого початку розроблено виключно для розробки додатків для браузеру на такій мові програмування як JavaScript, тому воно має всі необхідні інструменти для швидкої розробки та тестування програмного продукту. Стандартні інструменти вміють правильно доповнювати строки коду, що пише програміст, а також допомагає шукати та усувати помилки. Крім того у WebStorm можна додати велику кількість налаштувань, таких як ESlint, плагіни для препроцесорів CSS та інше, що допоможе значно покращити якість коду та швидкість розробки ПЗ.

Використовувати всі можливості WebStorm може кожен розробник, адже на офіційному сайті JetBrains доступна 30–денна безкоштовна версія IDE, яка покаже вам всі переваги саме цього середовища розробки та надасть вам час для того, щоб ви відточили свої навички.

На сьогодні WebStorm та PhpStorm – це комплексне вирішення для написання Web–застосунку любої складності, адже обидва вони надають якісну підтримку таких Web–технологій як:

— Angular;

— React;

— Vue.js.

Для мобільної розробки користувач цієї IDE може використовувати такі технології та фреймворки як:

— Ionic;

— Cordova;

— React Native.

Для серверної та десктопної розробки з повною підтримкою середовища можна використовувати найбільш популярні технології, такі як:

— Node.js;

— PHP;

— Meteor;

— Electron.

Такий набір технологій показує, що на сьогоднішній день JetBrains WebStorm у повній мірі відповідає всім тенденціям та новинкам у напрямі Web–розробки, та заслужено посідає перші місця серед середовищ розробки JavaScript та Node.js [15-17].

На момент написання дипломного програмного продукту використано актуальну версію WebStorm 2019.1.1, яка має такі особливості для технологій, обраних для розробки сервісу шифрування–дешифрування:

— додано можливість автоматично деструктуризувати значення із масивів JavaScript, використовуючи дуже стислий синтаксис. Просто натисніть Alt–Enter та упровадите деструктуризацію в код JavaScript чи TypeScript;

— тепер можна автоматично змінити функцію, яка повертає Promise за допомогою викликів .then() і .catch() до функції async, яка використовує синтаксис async/await. Просто натисніть Alt–Enter на ім’я функції та

оберіть «Перетворити на функцію async». Це можливо не тільки в файлах TypeScript, але і в JavaScript і Flow;

— тепер WebStorm використовує службу мови TypeScript разом зі своєю власною підтримкою TypeScript для будь–якого коду TypeScript у файлах .vue. Це означає, що тепер ви отримаєте більш точну перевірку типів і введіть інформацію, ви зможете використовувати швидкі виправлення, надані службою, і побачити всі помилки TypeScript у поточному файлі у вікні інструменту TypeScript;

— якщо ви використовуєте модулі CSS у вашому проекті, завершення коду для класів у файлі JavaScript тепер пропонуватиме версії клавіш з іменами класів з тире;

— за допомогою нового рефакторингу змінних CSS можна замінити всі використання значення поточного файлу .css на змінну за допомогою синтаксису var (– var–name).

Важливо не тільки реалізувати певний функціонал, а і зробити це якісно та попіклуватися про те, щоб розроблений сервіс однаково якісно працював у всіх популярних, та більшій частині другорядних браузерів, і з цим дуже гарно допомагає такий плагін, як ESLint для JavaScript та TSLint для TypeScript. Даний плагін завжди вкаже на недоліки, які були допущені програмістом та допоможе автоматично переформатувати програмний код під всі нові стандарти, та зробити його більш стислим та зрозумілим [13].

Середовище розробки WebStorm ідеально підтримує всі найновіші технології та фреймворки для мови JavaScript, а тому кожен розробник отримує прості інструменти доповнення коду та навігації в контексті завдяки розширеному синтаксису об’єктів, переходам GoTo та інформаційним меню Alt–Enter, яке надасть всю необхідну інформацію про клас, об’єкт чи структуру, яку ви використовуєте чи збираєтесь використати у своєму проекті. Крім цього варто відмітити зручний інтерфейс, який можна дуже гнучко налаштувати під себе.

Пошук по всьому проекту спрощується завдяки групованню, фільтрації і пошукові в результатах, а також дозволяє зберігати довільне число наборів результатів. Покращене великомасштабне структурне уявлення панелі

прокрутки дозволяє швидко знаходити проблеми, а візуалізація структури завжди підкаже, в якому місці блокової структури коду ви перебуваєте.

Значки лампочок навпроти вказівника строки покажуть, де ви допустили помилку, а також допоможуть її виправити. Для більшої зручності у стандартному інтерфейсі WebStorm зліва від скролбару кожен розробник має вертикальну строку з відмітками місця помилок і звичайно може швидко перенести фокус саме на це місце [12].

Починаючи з версії 2019.1.1 всі користувачі JetBrains WebStorm отримали ще більш зручний та багатофункціональний відладчик коду. Нова консоль має всі необхідні налаштування для JavaScript та Node.js та відображає об’єкти, використовуючи деревоподібний перегляд, підтримує укладання повідомлень журналу за допомогою CSS і групування їх за допомогою console.group() і console.groupEnd(). Ви також можете відфільтрувати будь–які повідомлення журналу.

Крім цього нова консоль додаванні скриптів до файлу package.json, WebStorm тепер надає пропозиції щодо доступних команд, що надаються встановленими пакетами. Після вводу вузла IDE запропонує імена папок і файлів. А після введення npm запуску відображатиметься список завдань, визначених у поточному файлі.

Виконайте налаштування інтерфейсу індивідуально під себе, опираючись на свої персональні вподобання, адже WebStorm має більш ніж 10 стандартних тем та навіть дозволяє завантажувати свої персональні плагіни. Крім кольорового оформлення ви можете налаштувати розташування всіх інструментів розробника в залежності від ваших вподобань та опираючись на зручність.

2.2 Вибір архітектури програмного комплексу

Для реалізації поставленої в дипломній роботі задачі прийнято рішення використовувати архітектуру, яка складається з трьох рівнів: сервер, база даних і клієнта. Схему даної архітектури наведено на рисунку 2.1 [13-15].

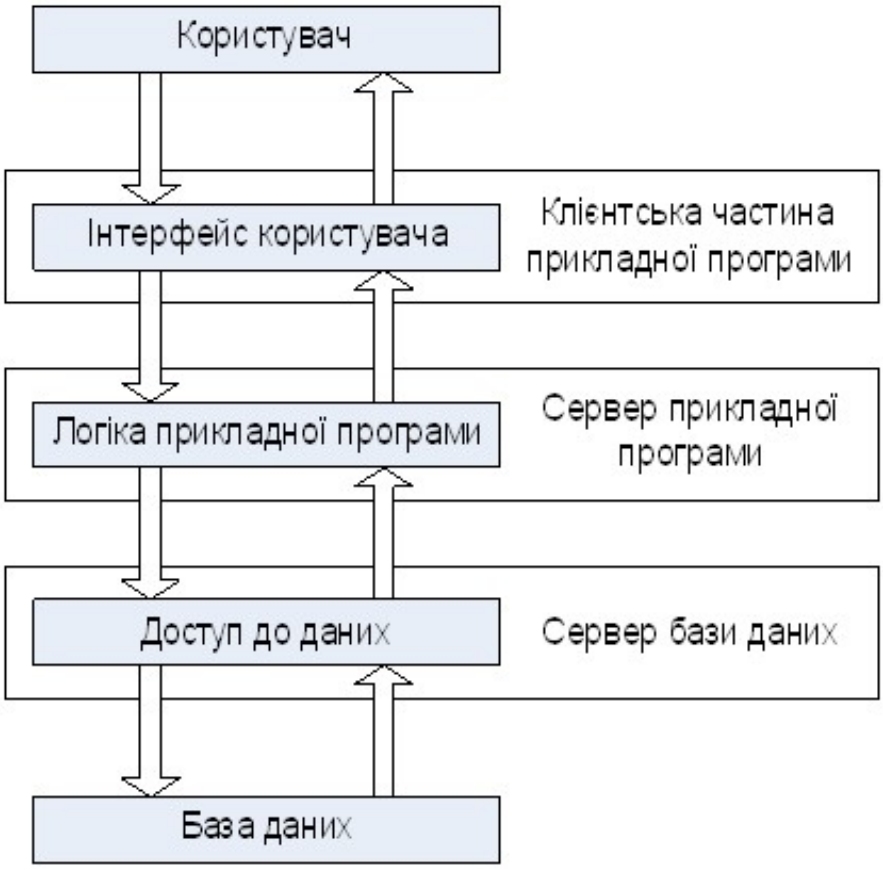

Рисунок 2.1 – Триланкова архітектура програмного комплексу

У програмуванні та проектуванні комп’ютерних систем триланкова архітектура (англ. three–tier або Multitier architecture) являє собою структуру програмної системи або додатку, який має наступний набір компонентів:

— клієнтський застосунок – інтерфейс користувача, підключений до сервера застосунків;

— сервер застосунків – ядро програми, яке представляє собою основну логіку та функціонал системи та взаємодіє як з інтерфейсом користувача, так і з сервером бази даних;

— сервер баз даних – цей рівень відповідає за збереження інформації, та її представлення для рівня серверу застосунків. Даний рівень потребує найбільшої захищеності, та окрім самих даних має схему та тригери, та процедури БД.

У найпростішому вигляді триланкова архітектура Web–додатку може буди розміщена на двох машинах, а саме – клієнтський застосунок, та на іншій – реалізація бази даних та серверна частина додатку.

У розширеному вигляді сервер бази даних необхідно переносити на окремий сервер або кластер для більшої надійності та для масштабування системи, адже до одного кластеру можуть бути підключені декілька терміналів або навіть серверних комп’ютерів [15-17].

Розглянемо основні переваги триланкової архітектури:

— можливість простого масштабування системи та її переносу на інші машини;

— ізоляція рівнів один від одного. Ця перевага дозволяє швидко та незалежно від інших рівнів проводити оновлення та конфігурування того чи іншого рівня системи;

— висока безпека;

— висока надійність;

— не обов’язковою є висока швидкість передачі даних між сервером БД та терміналами;

— більш низька вартість комплектуючих клієнтського комп’ютеру, адже основну продуктивність системи сконцентровано на серверній частині та у БД.

Всі недоліки такої архітектури витікають з переваг, а саме варто відзначити такі пункти:

— більша складність розробки програмної системи (більша кількість використаних мовних рішень, фреймворків та патеренів проектування комп’ютерних систем);

— важчий порядок для розгорнення системи та її підтримки системними адміністраторами;

— дорожчі комплектуючі для сервеної частини системи та для розроблюваної БД;

— необхідність високої швидкості передачі даних між сервером та клієнтськими додатками.

Саме ця концепція взаємодії була використана в першу чергу для того, щоб розділити навантаження між учасниками взаємодії та процесу обміну інформацією, а також для того, щоб розділити програмний код та зробити його більш зрозумілим та структурованим. Такий підхід є гарним рішенням як з точки зору програмування, так і у плані зручності представлення програмного продукту кінцевому споживачеві в цілому.

2.3 Опис архітектури серверного додатку

Для проектування серверної частини додатку використовувався шаблон проектування MVC (Model View Controller), структуру якого проілюстровано на рисунку 2.2 [17].

Цей шаблон пропаганду підхід до розробки програмного забезпечення, прі якому основна логіка розподіляється на три окремі функціональні ролі: модель даних (model), користувацький або графічний інтерфейс (view) і керуючу логіку (controller). Таким чином, окремі зміни в тому чи іншому рівні не мають ніякого відношення до інших рівнів та мають мінімальний вплив та незначні відношення між собою [17-19].

Розглянемо основні структурні ланки даного патерну розробки програмного забезпечення:

— модель (Model) – забезпечує об’єктну модель певної предметної області, включає дані та методи роботи з цими даними, відповідає на запити від контролера, повертає дані та/або змінює його стан, а модель не містить даних, подібних даних. Ви можете візуалізувати, а також не "спілкуватися" з користувачем безпосередньо;

— презентація (View) – відповідає за відображення інформації (візуалізації), однакові дані можуть бути представлені різними способами, наприклад, колекція об’єктів, що використовують різні «види», може бути представлена у вигляді таблиці або списку;

— контролер (Controller) – забезпечує зв’язок між користувачем і системою, використовує модель і перегляд для реалізації необхідної

реакції на дії користувача, як правило, на рівні контролера, отримані дані фільтруються і авторизуються (права користувача на виконувати дії або отримувати інформацію перевіряються).

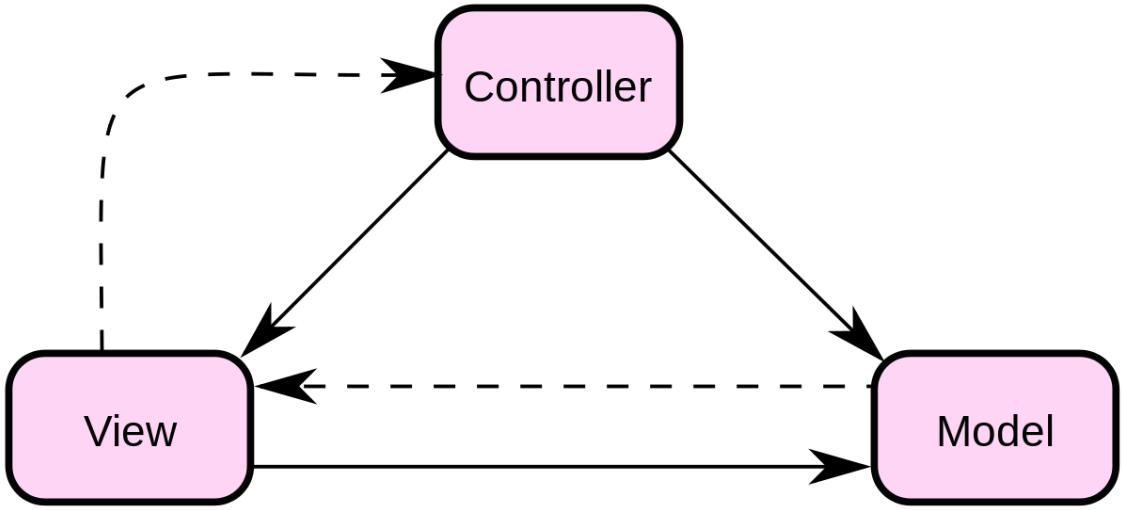

Рисунок 2.2 Паттерн проектування MVC – Model View Controller

Розглянемо детальніше патерн MVC у такому вигляді, як його застосовано у дипломному проекті.

Контролер – це середовище, яке в першу чергу відповідає за доступ до даних, тобто саме у ньому виконуються POST та GET запити до серверу бази даних та виконується маршрутизація серверу. Окрім запитів на рівні контролера виконується первинна обробка змінних оточення, їх наявність та визначення значень по замовченню. Також цей рівень відповідає за запис та відстеження помилок, якщо роботу програми буде порушено.

Модель – виконує повноцінні перевірки параметрів, від яких залежіть коректність роботи програми (перевіряються діапазони, допустимість використання змінних і так далі). Також варто відмітити, що саме підключення та взаємодія з базою даних та іншими сховищами виконується також на рівні моделі додатку. Останньою задачею є підготовка даних до візуалізації даних та передачі їх на рівень представлення [18, 19].

Представлення – найбільш тонкий рівень в плані логіки, адже на ньмо ми лише отримужмо та виводимо дані користувачеві. У розроблюваній системі у

якості представлення використовувався фреймворк для мови JavaScript – Vue.js, та бібліотека Nuxt.js.

Розглянемо основні недоліки та переваги застосування патерну MVC. Спочатку розглянемо основні недоліки, які виявлено в ході розробки системи: 1. Необхідність використання більшої кількості ресурсів. Складність є наслідком того, що всі три основні блоки повністю незалежні і діють виключно з передачею даних. Контролер повинен завжди завантажувати (і при необхідності створювати) всі можливі комбінації змінних і переносити їх на модель. Модель повинна завантажити всі дані візуалізації та переслати їх на дисплей. Наприклад, в модульному підході модуль може безпосередньо обробляти змінні середовища і візуалізувати дані, не завантажуючи їх в окремі частини пам'яті; 2. Комплексний механізм розподілу програм на модулі. У концепції MVC жорстко написано наявність трьох блоків (Model, View, Controller). Відповідно, кожен функціональний модуль повинен складатися з трьох блоків, що в свою чергу дещо ускладнює архітектуру функціональних модулів програми; 3. Складний процес розширення функціональності. Проблема дуже схожа на описану вище. Не досить написати функціональний модуль і зв'язати його з одним місцем програми. Кожен функціональний модуль повинен складатися з трьох частин, кожен з яких повинен бути з'єднаний з відповідним блоком [19, 20].

Тепер розглянемо переваги концепції MVC:

1. унікальна концепція системи. Неминучою перевагою MVC є унікальна глобальна архітектура додатків. Навіть у складних системах розробники (подібно до тих, хто розробив систему і які приєдналися до системи) можуть легко переміщатися за допомогою програмних блоків. Наприклад, якщо виникає помилка логіки даних, програміст негайно відхиляє 2–блок програму (контролер і перегляд) і займається третім (модельним) дослідженням. Я був здивований, наскільки сильно спрощена локалізація проблеми;
2. спрощений процес видалення помилок програми. Таким чином, весь механізм візуалізації тепер зосереджений в одному програмному блоці, а механізми для додаткового виведення графічних елементів спрощуються.

Я не можу зрозуміти, як ця претензія стосується програмування класичних додатків, але в Web програмуванні ця архітектурна особливість стала безсумнівним перевагою.

2.4 Опис архітектури клієнтського додатку

Як відомо, за останні 10 років мова JavaScript стала значно більш зрілою. Більшість коду, який раніше виконувався на стороні сервера, мігрував на сторону клієнта і тепер працює в браузерах. Так як код став більш складним, JavaScript фреймворки були необхідні, щоб допомогти структурувати цей код і реалізувати простішу підтримку клієнтів.

Метою створення Vue.js є забезпечення простих у вивченні, універсальних, потужних, простих у обслуговуванні та тестування JavaScript–фреймворків. Vue.js також прагне бути прогресивним. Це означає, що якщо у вас вже є готовий проект, то ви можете легко додати підтримку Vue до цього проекту, розширюючи таким чином функціональність та інтерактивність існуючої програми. На даний момент Vue.js є єдиним фреймворком та user interface бібліотекою, яка може працювати та застосовуватися даним способом та з даною ціллю [20-22].

Технічно Vue.js фокусується на шарі шаблону MVVM ViewModel, схема роботи якої зображена на рисунку 2.3. Підключається до View та Model з використанням двох методів прив’язки даних. Поточне маніпулювання DOM і форматування виводу скорочуються в директивах і фільтрах.

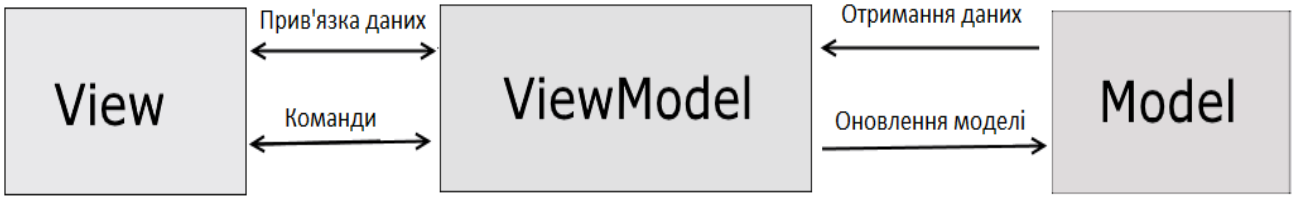

Рисунок 2.3 – Схема роботи MVVM шаблону

З точки зору представлення Vue.js використовує шаблон на основі DOM. Кожен екземпляр Vue асоціюється з відповідним елементом DOM. Коли створюється екземпляр Vue, він рекурсивно проходить всі дочірні вузли свого кореневого елемента, встановлюючи необхідні посилання на дані. Після компіляції дисплея він стає реагуючим на зміни даних.

Коли ви використовуєте Vue.js, вам рідко доводиться торкатися DOM, за винятком налаштованих директив. Оновлення стану додатку буде автоматично активовано при зміні даних. Всі оновлення надсилаються і виконуються асинхронно для більшої ефективності.

Шаблони Vue.js – це просто об’єкти JavaScript або об’єкти даних. Коли об’єкт використовується як дані в екземплярі Vue, він стає реактивним. Ви можете маніпулювати їхніми властивостями, і шаблони Vue будуть повідомлені про зміни. Vue.js досягає прозорої реактивності шляхом перетворення властивостей на об’єкти даних в ES5, які мають своє геттери та сеттери. Немає необхідності в постійній перевірці, і вам не потрібно явно сигналізувати Vue для оновлення дисплея. Коли дані змінюються, перегляд оновлюється в наступному вікні [23-25].

З філософської точки зору, мета полягає в тому, щоб забезпечити якомога простіші переваги реактивних зв’язків і складових частин API. Це не повний перелік переваг адже Vue призначений для легкого та гнучкого перегляду створення універсальних додатків найпростішим способом. Ви можете використовувати його для швидкого створення додатків або для використовувати разом з іншими бібліотеками для формування спеціального стека розроблюваного додаттку.

Програмний інтерфейс API Vue.js значно залежить від AngularJS, KnockoutJS, Ractive.js і Rivets.js. Незважаючи на схожість, вважається, що Vue.js пропонує відмінну альтернативу існуючим бібліотекам, завдяки своїй простоті та функціональності.

2.5 Опис інструментів розробки

Програмний комплекс базується на принципах триланкової програмної архітектури. Кожен рівень цієї архітектури реалізується за допомогою набору технологій, і основною метою є створення мультиплатформного рішення з використанням відкритих технологій.

Для клієнтського рівня використовувався наступний набір технологій: мова програмування JavaScript, реактивний веб–фреймворк Vue.js, набір компонентів для створення користувацьких інтерфейів та графічного дизайну – Vuetify.js.

Сучасний JavaScript – це «безпечна» мова програмування загального призначення. Вона не забезпечує низький рівень доступу до пам’яті, процесору, тому що спочатку вона була орієнтований на веб–браузери, які цього не потребують.

Що стосується інших функцій – вони залежать від середовища, в якому запущений JavaScript [18]. Браузер разом з JavaScript може робити все, що стосується маніпулювання сторінкою, взаємодії з відвідувачем і, певною мірою, сервером:

— створювати нові HTML–теги, видаляти існуючі, змінювати елементи стилю, приховувати та показувати елементи, тощо;

— реагувати на дії відвідувачів, обробляти кліки миші, переміщати курсор, натискати на клавіатуру і так далі;

— надсилати запити на сервер і завантажувати дані без перезавантаження сторінки (ця технологія називається «AJAX»);

— можна отримувати та встановлювати файли cookie, шукати інформацію та відображати повідомлення.

Vue.js – це прогресивна основа для створення користувацьких інтерфейсів. На відміну від монолітних фреймворків, Vue призначений для поступового розширення. Ядро Vue в першу чергу вирішує завдання презентації (перегляду), що робить більш простою інтеграцію з іншими фреймворками, бібліотеками та існуючими проектами. З іншого боку, Vue також ідеально підходить для

створення Single–Page Applications (SPA), якщо використовується разом з сучасними інструментами та додатковими бібліотеками. Vue дотримується компонентного підходу до розробки. Компоненти розробляють ідею плагінів. Кожен з них досягає деяких своїх здібностей (а якщо їх немає, то можна написати власну). Якщо вам потрібно реалізувати компонент в іншому місці, його можна легко використовувати повторно. Взаємодію можна описати за допомогою простого інтерфейсу: ми надсилаємо вхідні параметри в конструктор компоненту, а для зворотного зв'язку можна відстежувати події [23-27].

Все це стосується компонентів. З тією лише різницею, що компонент може бути не тільки одним (наприклад, функціональним вибором), але і частиною програми, яка повинна працювати і виглядати скрізь однаковим чином (наприклад, форма коментарів, з аватарами).

Використання компонентів дозволяє уникнути дублювання коду і чітко побудувати архітектуру програми. Кожну складну сторінку завжди можна розділити на менші компоненти. Кожну з цих частин в розподільчому компоненті легше підтримувати, і при необхідності повторювати розщеплення в межах компонента на ще менші частини.

Першою вагомою перевагою такого поділу компонента є зручність підтримки – вам більше не потрібно описувати логіку всієї програми глобально, ви можете зосередитися на певній її частині. Зміни або вдосконалення потрібні лише в одному місці. Ізоляція компонентів усуне конфлікти з іншими частинами програми.

Тому використання компонентних підходів зараз широко використовується в багатьох JavaScript бібліотеках. Vue стоїть поруч з найбільш популярними рішеннями для написання web–застосунків і забезпечує відмінні можливості для роботи з компонентами.

Сьогодні Vue.js активно набирає популярність, в доведення цього варто відмітити, що Vue використовується для написання офіційного сайту Apple.

2.6 Універсальні додатки та серверний рендеринг

Бібліотека Nuxt.js – це фреймворк високого рівня, який додається до Vue. Він спрощує розробку універсальних додатків або окремих сторінок веб-сайтів, написаних на Vue.

Модуль Nuxt.js пропагандує розподіл коду сервера і клієнта, щоб можна було зосередитися на розробці додатків. Мета Nuxt – бути достатньо гнучкою платформою, щоб використовувати її як основну базу проекту. Більшість того, що робить Nuxt на етапі розробки, ви отримуєте у якості переваг лише за допомогою декількох додаткових файлів, доданих до вашої директорії файлів.

Давайте розглянемо причини, чому варто використовувати Nuxt разом з проектом Vue. Однією з найбільших переваг в Nuxt.js є те, що Nuxt дозволяє легко створювати універсальні програми. Що таке універсальна програма? Універсальний додаток використовується для опису коду JavaScript, який може бути виконаний на клієнті і на стороні сервера.

Багато сучасних фреймворків JavaScript, в тому числі і Vue, призначені для створення програми з одним сайтом (SPA). Є багато переваг для того, щоб мати не SPA, а традиційний веб–сайт. Наприклад, ви можете створювати дуже швидкі інтерфейси користувача, які швидко оновлюються. SPA мають ряд недоліків, таких як довгий час завантаження, і Google бореться з ними, тому що немає змісту, через що виникають проблеми з SEO, адже весь зміст генерується завдяки JavaScript після завантаження скриптів [25-28].

Універсальне додаток на основі SPA можно розробити за допомогою Nuxt, але замість порожньої сторінки index.html, попередньо встановіть програму на веб–сервері і надішліть HTML–розмітку у відповідь на запит браузера для кожного маршруту, щоб прискорити час завантаження і покращити SEO, що полегшує пошуковим системам, таким як Google сканувати сайти.

Це проблема, яку Nuxt.js хоче вирішити для додатків Vue. Nuxt.js полегшує обмін клієнтським і серверним кодом, щоб можна було зосередитися на логіці програми.

Nuxt.js надає вам доступ до таких функцій, як isServer і isClient на ваших компонентах, так що ви можете легко вирішити, чи слід відображати щось на клієнті або на сервері. Існують також спеціальні компоненти, такі як компонент «no–ssr», який використовується для навмисного запобігання відображенню компонентів сервера.

Нарешті, Nuxt надає доступ до методів asyncData у компонентах, які можна використовувати для отримання даних і подачі на стороні сервера.

Це вершина айсберга про те, як Nuxt допомагає створювати універсальні програми.

2.7 Дизайн клієнтського застосунку

Дизайн є важливим для сайту, оскільки він є «лицем» компанії, або навіть брендом. У той же час гарний дизайн не обов'язково повинен бути складним і яскравим, в тенденції - простота і мінімалізм. Важливо реагувати на дух компанії, її фірмовий стиль, якщо такий є, на образ компанії в цілому. Приємно бути в прекрасному місці, і користувачі з великою симпатією відносяться до ресурсів, які створюють почуття краси та зручності. Гарний дизайн - це повага до користувача.

Для проектування та реалізації дизайну дипломного проекту застосовано бібліотеку компонентів для Vue, яка має назву Vuetify, та відповідає всім стандартам Material Design, описаних на офіціальному сайті Google. Всі ці компоненти написані великою спільнотою програмістів, підтримують SSR, та легко впроваджуються у любий додаток. Бібліотека доступна для вільного використання на GitHub і є одним із найкращих виборів для створення якісного та стандартизованого інтерфейсу користувача.

В релізі v 1.0 alfa для розробників доступно більш ніж 80 компонентів, що підходять для повторного використання і спроектованих із застосуванням семантичних принципів і простих назв.

2.8 Інструменти розробки серверної частини додатку

Платформа Node.js – серверна мова програмування, яка має синтаксис мови JavaScript та дозволяє швидко розроблювати та легко масштабувати серверні додатки. Ця мова програмування була розроблена в 2009 році та одразу отримала велику прихильність. Node.js використовує керовану подіями модель вводу–виводу, що робить його надзвичайно ефективним і робить можливим масштабоване мережеве застосування.

Маючи більше мільярда завантажень, Node.js процвітає у створенні програм у реальному часі, Інтернету речей та мікросервісах. Він набирає обертів швидше, ніж будь–яка інша технологія, і займає одне з найвищих навичок розробника [28-30].

Найважливіші переваги Node:

— можливість створення додатків у реальному часі (наприклад, чат або ігри) швидким;

— цю програмування JavaScript можна використовувати для кодування в як для клієнта, так і для сервера;

— підвищує ефективність процесу розробки, оскільки заповнює розрив між розробниками frontend та backend;

— постійно зростаючий NPM (Node Package Manager) дає розробникам багато інструментів і модулів для використання, тим самим підвищуючи їх продуктивність;

— код виконується швидше, ніж на будь–якій іншій мові;

— Node ідеально підходить для мікросервісів, які є популярним рішенням серед корпоративних додатків.

Кожна мова програмування дасть вам кілька причин вибрати їх серед інших. Перевага в Node.js полягає в тому, що він призначений для застосування у додатках, які планується розширювати. Сучасні інструменти та великі перспективи роблять Node.js одною із самих популярних та використовуваних технологій програмування.

Один вузол архітектури Node.js, який ініціює подію, дозволяє ефективно керувати кількома одночасними з'єднаннями. Більшість популярних веб–платформ створюють додаткові потоки для кожного нового запиту, використовуючи оперативну пам'ять весь час, необхідний для обробки. Вузол, з іншого боку, працює на одному потоці, використовуючи цикл подій і зворотній зв'язок для операцій вводу/виводу, видаляючи завдання як операцію бази даних якомога швидше. Це дозволяє обробляти сотні тисяч або навіть мільйони одночасних з'єднань.

Це фактично означає, що Node.js не є новою «золотою» платформою, яка буде домінувати у світі веб–розробки. Натомість, це платформа, яка відповідає певній потребі. Розуміння є абсолютно необхідним. Ви, безумовно, не хочете використовувати Node.js для CPU–інтенсивних операцій; насправді, його використання для важких розрахунків скасовує практично всі її переваги. Але Node дійсно поєднує в побудові швидких, масштабованих мережевих додатків, оскільки він може керувати великою кількістю високошвидкісних високошвидкісних з'єднань, які дуже масштабовані.

Цікаво, як він працює під капотом. У порівнянні з традиційними методами веб–сервісів, коли кожне підключення (запит) генерує новий потік, видаляє систему ОЗП і, нарешті, максимальну кількість оперативної пам'яті, Node.js працює в одному потоці за допомогою розблокування/викликів, які підтримують десятки тисяч одночасних посилання в циклі подій.

Для розвернення серверу використовувався фреймворк Express for Node.js. Написаний на JavaScript, Express діє лише як тонкий шар основних функцій веб–додатків. На відміну від великої, дуже продуманої рамки, як Ruby on Rails, Express не має об'єктного реляційного відображення або двигуна шаблонів. Експрес не побудований навколо конкретних компонентів, маючи «немає думки» про те, які технології ви підключаєте до нього. Вона прагне поставити управління в руки розробника і полегшити розробку веб–додатків для Node.js. Ця свобода, в поєднанні зі швидким налаштуванням і чистою середовищем JavaScript Node, робить експрес сильним кандидатом для швидкого розвитку та швидкого прототипування. Express користується найбільшою популярністю у

стартапів, які хочуть побудувати продукт якомога швидше і не мають багато застарілого коду.

2.9 Проектування бази даних

MongoDB – це об'єктно–орієнтована, проста, динамічна і масштабована база даних NoSQL. Вона базується на передачі моделі документами NoSQL. Об'єкти даних зберігаються як окремі документи в бібліотеці – замість зберігання даних у стовпцях і рядках традиційної реляційної бази даних. Мотивацією для мови MongoDB є реалізація сховища даних, що забезпечує високу продуктивність, високу доступність і автоматичне масштабування. MongoDB дуже простий в установці і розгортанні. MongoDB для зберігання використовує документи JSON або BSON. Порівняння основних «термінів» бази даних SQL NoSQL наведено на рисунку 2.4.

SQL Server	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Column	Field
Joining	Linking & Embedding
Partition	Sharding (Range Partition)
Replication	ReplSet

Рисунок 2.4 – Порівняння термінології SQL Server та MongoDB

Розглянемо плюси і мінуси застосування бази даних Mongo. До переваг варто віднести такі пункти:

— документ-орієнтований підхід (рисунок 2.5);
— висока ефективність;
— висока доступність – реплікація;
— висока масштабованість;
— динамічна – немає жорсткої схеми;
— гнучке додавання / видалення поля не впливає на додаток;

— гетерогенні дані;

— немає з’єднання;

— розподілений;

— представлення даних у JSON або BSON;

— геопросторову підтримку;

— легка інтеграція з BigData Hadoop;

— мова запитів на основі документів, яка є майже такою ж потужною, як SQL;

— хмарні розподіли, такі як AWS, Microsoft, RedHat, dotCloud і SoftLayer тощо: Фактично, MongoDB побудований для хмари. Його власна архітектура масштабування, включена за допомогою «sharding», добре узгоджується з горизонтальним масштабуванням і гнучкістю, що забезпечується хмарними обчисленнями.

Головними недоліками є:

— Недоліком NoSQL є те, що більшість рішень не настільки сильно сумісні з ACID (Atomic, Consistency, Isolation, Durability), як більш добре встановлені системи RDBMS;

— Складне зв’язування моделей;

— Не існує жодної функції або збереженої процедури, де можна прив’язати логіку.

MongoDB 4.0 додав підтримку для декількох транзакцій документів, що робить його єдиною базою даних, яка поєднує гарантії ACID традиційних реляційних баз даних, швидкість, гнучкість і потужність моделей документів та інтелектуальних розподілених систем – для масштабування та налаштування, де це необхідно. Завдяки ізоляції знімків транзакцій, вони забезпечують послідовне відображення даних і забезпечують виконання будь–яких дій або нічого для збереження цілісності даних. Операції MongoDB відчувають себе так само, як розробники знайомі з транзакціями в MySQL. Він складається з декількох операторів з подібним синтаксисом (наприклад, start_transaction і commit_transaction), і тому будь–який користувач з попереднім досвідом транзакцій може легко додати будь–яку програму [28-30].

На відміну від MySQL та інших реляційних баз даних, MongoDB побудований на архітектурі розподілених систем, а не на монолітному, одному вузлі. Як результат, MongoDB пропонує нестандартне масштабування та локалізацію даних за допомогою автоматизованого керування та реплік для підтримки постійної доступності.

В цілому, підсумовуючи варто відмітити, що MongoDB – це база даних загального призначення, яка використовується для різних випадків використання та має структуру, подібну до тої, яку зображено на рисунку 2.5. Найбільш поширеними випадками використання MongoDB є Single View, Інтернет речей, мобільний, аналітика в реальному часі, персоналізація, каталог і управління вмістом. З додаванням транзакцій з декількома документами, Вам ще простіше звертатися до повного спектру випадків використання з MongoDB.

Рисунок 2.5 – Приклад структури таблиці бази даних MongoDB.

Висновки до розділу

Для розробки Web–сервісу шифрування–дешифрування даних прийнято рішення використовувати набір фреймворків мови програмування JavaScript, таких як Vue.js, Nuxt.js – для клієнтського додатку та Node.js – для серверного та базу даних MongoDB. Такий набір технологій разом з патерном проектування MVC дозволяє в майбутньому легко розширювати функціонал системи та робити її небезпечною та захищеною від зловмисників.

3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблений проект має наступну структуру:

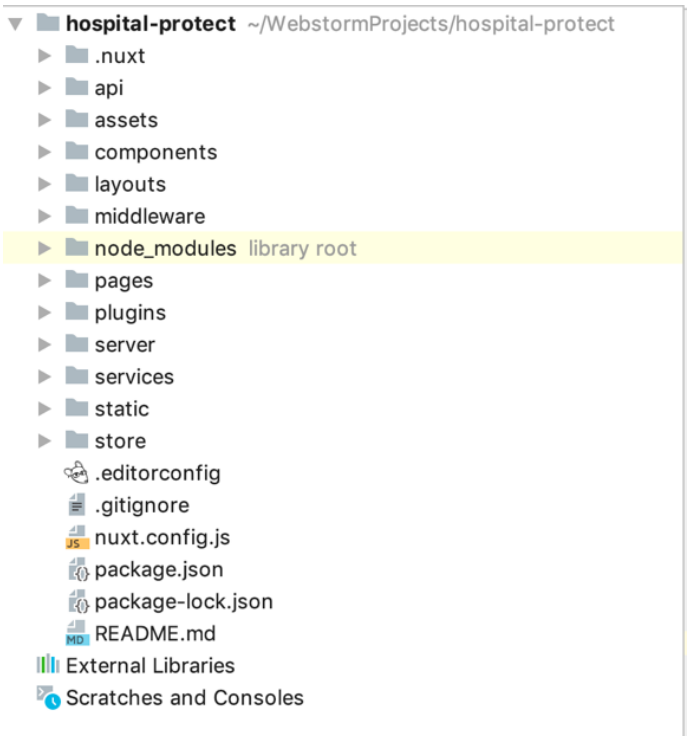

Рисунок 3.1 – Ієрархія папок проекту

Основними складовими частинами є піддиректорії pages, components, api, store.

— pages – стандартна директорія Nuxt, яка є основною для реалізації frontend частини, адже у ній знаходяться всі основні шаблони для сторінок, а також завдяки цій директорії налаштовується маршрутизації проекту. У нашому випадку структура папки pages має вигляд, зображений на рисунку 3.2;

— папка components – це основні складові сторінок, які мають таку ж структуру, як і компоненти з директорії pages, але не мають методу асинхронних даних asyncData();

— layouts – шаблони сторінок, які часто використовуються у додатку

— каталог Middlewarem – проміжне програмне забезпечення, яке містить методи, які виконуються при певних умовах. Проміжне програмне

забезпечення дозволяє визначати користувальницькі функції, які можна запускати перед відображенням сторінки або групи сторінок (макетів).

— каталог plugins містить ваші додатки Javascript, які потрібно запустити перед створенням кореневого додатка Vue.js. Це місце для глобального реєстрації компонентів і введення функцій або констант.

— каталог Store містить ваші файли Vuex Store. Vuex Store поставляється з Nuxt.js з коробки, але за замовчуванням вимкнено. Створення файлу index.js у цьому каталозі дозволяє активувати сховище і почати працювати з ним;

— файл nuxt.config.js містить спеціальну конфігурацію Nuxt.js;

— файл package.json містить залежності та сценарії програми.

Рисунок 3.2 – Структура папки pages

З компонентів, які знаходяться саме у папці pages виконуються запити на сервер, і саме ці компоненти передають дані на компоненти «без логіки», які знаходяться у папці components, та служать у більшості контентом для компонентів вищого порядку, описаних у pages.

3.1 Опис таблиць бази даних

База даних системи складається з чотирьох таблиць не реляційної бази даних, які створюють єдиний інформаційний простір для зберігання та отримання доступу до даних.

Першою таблицею є «hospital–crypto» (рисунок 3.3) – таблиця, яка зберігає унікальні ключі для кожного користувача, які генеруються за вищеописаними формулами та присвоюються індивідуально до користувача у зашифрованому вигляді та використовуються для шифрування–дешифрування даних, які стосуються кожного користувача системи.

Таблиця має таку структуру:

```
let Post = new Schema({
  id: {
    type: String
  },
  x: {
    type: String
  },
  y: {
    type: String
  },
  k: {
    type: String
  },
  alfa: {
    type: String
  }
},{
  collection: 'hospital-crypto'
});
```

Рисунок 3.3 – структура таблиці hospital–crypto

Друга таблиця – таблиця пацієнтів лікарні – «hospital–patient», яка зберігає інформацію про кожного пацієнта і має таку структуру (рисунок 3.4)

```
let Post = new Schema({
  status: {type: String},
  firstName: {type: String},
  secondName: {type: String},
  gender: {type: String},
  birthDate: {type: String},
  doctor: {type: String},
  email: {type: String},
  phone: {type: String},
  password: {type: String}
},{
    collection: 'hospital-patients'
});
```

Рисунок 3.4 – структура таблиці hospital–patient

Третя таблиця – таблиця лікарів – «hospital–doctors», яка зберігає інформацію про кожного лікаря і має таку структуру (рисунок 3.5)

```
let Post = new Schema({
    status: {type: String},
    firstName: {type: String},
    secondName: {type: String},
    gender: {type: String},
    birthDate: {type: String},
    email: {type: String},
    phone: {type: String},
    profession: {type: String},
    category: {type: String},
    password: {type: String}
},{
    collection: 'hospital-doctors'
});
```

Рисунок 3.5 – структура таблиці hospital–patient

Четверта таблиця – «hospital–diseases» (рисунок 3.6) – таблиця, яка зберігає інформацію про захворювання пацієнтів, і має таку структуру:

```
let Post = new Schema({
    userId: {type: String},
    diagnosis: {type: String},
    description: {type: String},
    treatment: {type: String},
    date: {type: String}
},{
    collection: 'hospital-diseases'
});
```

Рисунок 3.6 – структура таблиці hospital–patient

3.2 Розробка кабінету користувача

Розроблена система передбачає реєстрацію та логін користувачів, а тому в ході проектування і розробки додатку було прийнято рішення реалізувати даний модуль за допомогою вже існуючих інструментів, а саме Google Firebase Authentication module. Даний програмний модуль дуже простий в налаштуванні та надає весь необхідний функціонал для створення, реєстрації та авторизації користувачів.

Сервіс Firebase надає не тільки зручне API для роботи з системою, а і зручну та інтуітивно понятну адміністративну панель управління, яку зображено на рисунку 3.7.

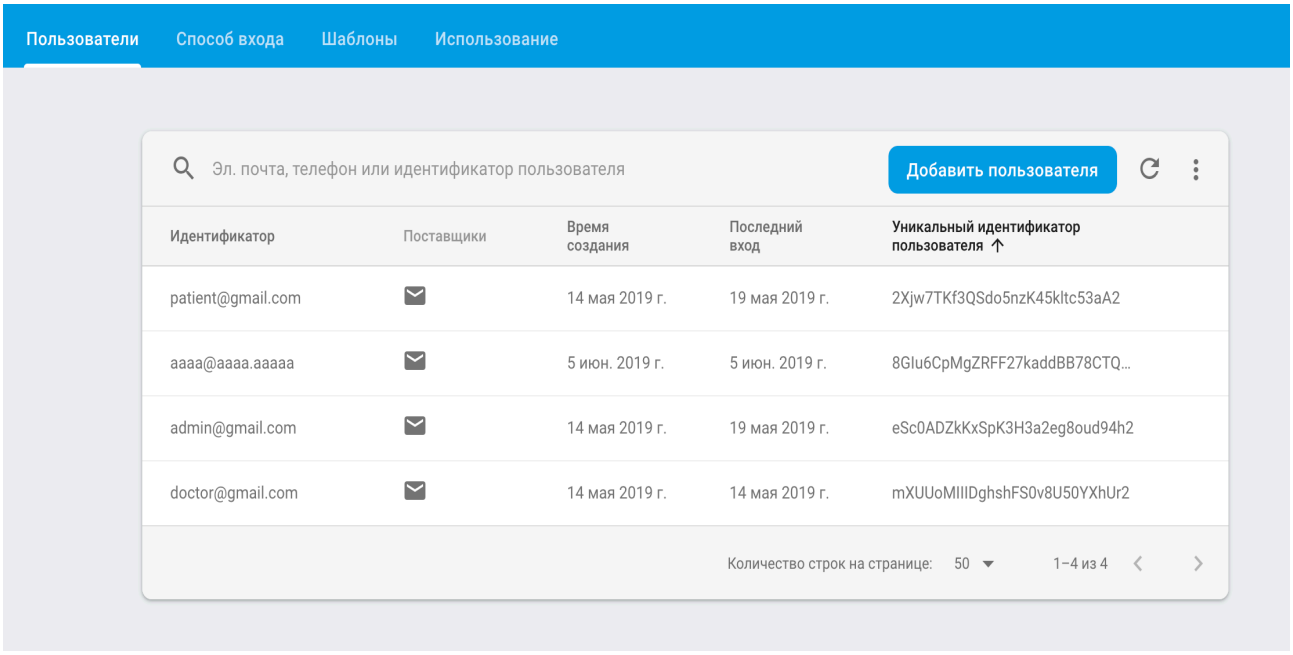

Рисунок 3.7 – інтерфейс адміністративної панелі Firebase Auth

Система Firebase також надає анонімну аутентифікацію, яка зазвичай використовується для зберігання невеликих обсягів даних під час очікування клієнта для аутентифікації за допомогою методу постійної авторизації. Ці анонімні сеанси можуть бути налаштовані на тривалість у декілька днів, тижнів, місяців, навіть років, поки користувач не ввійде в систему постійно або очистить кеш браузера.

Веб-програми часто використовують локальні бази даних, такі як sessionStorage або localStorage або Cookies для виконання подібних цілей, що достатньо небезпечно, адже всі дані зберігаються на комп’ютері користувача.

Firebase надає серверні послуги, прості у використанні пакети розробки та бібліотеки інтерфейсів, готові до використання для аутентифікації користувачів різних додатків на всіх платформах. Аутентифікацію можна здійснити за допомогою паролів та інтегрованих систем ідентифікації - Google, Facebook, Twitter та ін. Це значно спрощує процес застосування та забезпечує надійний захист.

Ми будемо використовувати декілька ролей, оскільки користувачі можуть мати більше однієї ролі в додатку або системі. Наприклад, користувачем може бути адміністратор, доктор чи звичайний користувач. Призначати властивість ролі користувачам необхідно коли вони будуть створюватися в базі даних реального часу при реєстрації, а вже в Mongo будуть зберігатися ключі для шифрування/дешифрування персональних даних.

Кабінет користувача – це основне робоче місце користувача в системі. Він включає в себе інші підмодулі, які виконують основні функції системи, та є елементом компонування в системі.

Кабінет користувача має різний набір функцій в залежності від статусу користувача, під яким здійснено вхід у систему (1 – для пацієнта, 2 – для лікаря, 3 – для адміністратора)

Підмодулі кабінету користувача для пацієнту (статус 1):

— Модуль редагування даних користувача;

— Модуль перегляду захворювань та діагнозів;

Підмодулі кабінету користувача для лікаря (статус 2):

— Модуль редагування даних користувача;

— Модуль перегляду пацієнтів, закріплених за лікарем;

— Модуль перегляду захворювань по кожному пацієнту;

— Модуль редагування історії захворювань.

Підмодулі кабінету користувача для адміністратора (статус 3):

— Модуль редагування даних користувача;

— Модуль перегляду та редагування штату співробітників лікарні;

— Модуль перегляду та редагування пацієнтів клініки;

— Модуль редагування історії захворювань кожного пацієнта.

Висновки до розділу

В даному розділі дано опис програмної реалізації системи шифрування даних методом Ель-Гамаля, та наведено конкретні приклади того, як виглядає структура проекту. Для авторизації використано модуль Google Firebase, який дозволяє безпечно реєструвати та зберігати дані користувача та легко та зручно реалізовувати підтримання користувацької сесії.

4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблена програмний комплекс розроблений з використанням веб–технологій і тому працює в браузерах, які підтримують актуальні веб–стандарти.

4.1 Інсталяція та системні вимоги

Для коректного функціонування програмної системи необхідно забезпечити наступну апаратно-програмну конфігурацію, та переконатися, що серверна та клієнтська машина відповідає наступним мінімальним технічним вимогам:

— операційна система Windows 7, 8, 10, MacOS 10+, Linux Ubuntu 10+;

— 2 Гб оперативної пам’яті;

— 100 Мб вільного місця на жорсткому диску;

— встановлений на комп’ютері Node.js (версії 9+) та npm.

Оскільки застосунок працює з використанням веб–технологій, тому від не потребує встановлення на пристрій користувача. Проте, для використання необхідний веб–браузер, який підтримує актуальні веб–стандарти.

Для встановлення застосунку на стороні серверу необхідно на дистрибутивному диску знайти директорію «Information security system», відкрити її за допомогою терміналу та у корені директорії виконати команду npm install для завантаження програмних залежностей для клієнтського застосунку. У терміналі розпочнеться процес встановлення залежностей та пакетів (рисунок 4.1), те ж саме проробити у папці api, для встановлення пакетів та залежностей для застосунку, який відповідає за серверну частину додатку. Для запуску бази даних необхідно завантажити пакет MongoDB, встановити його, та слідувати інструкціям налаштування файлів для зберігання даних на сервері.

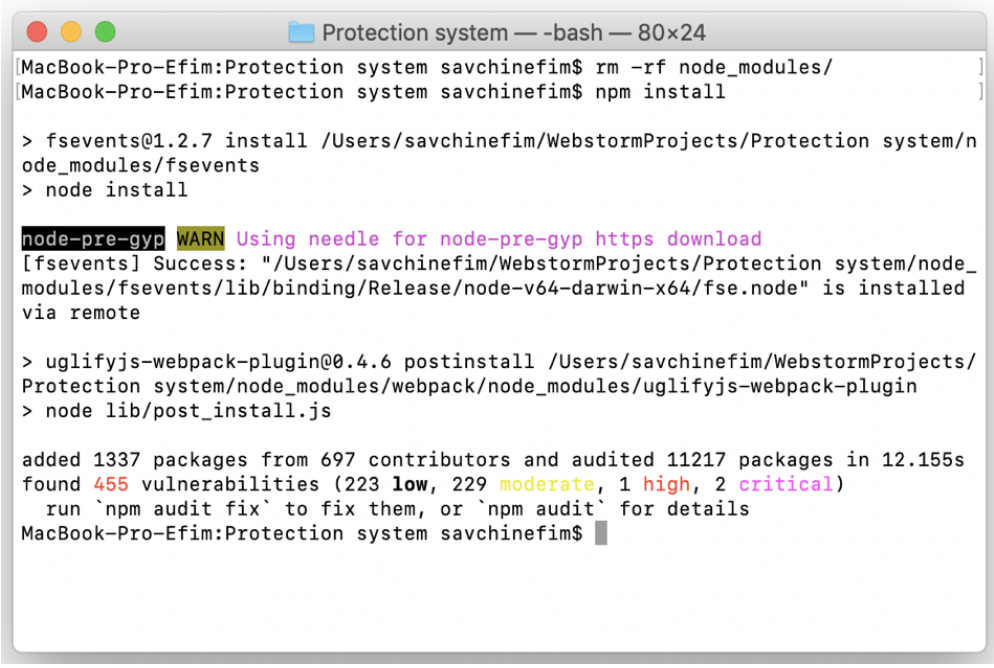

Рисунок 4.1 – Завантаження програмних залежностей та пакетів

Після завантаження залежностей виконайте команду «node server.js» у папці api та «npm run dev» у корені проекту.

Для отримання програмного коду додатку, готового до релізу на хостингу перенесіть папку «Information security system» на сервер та, виконайте команду npm install, а потім виконайте команду npm run build, яка запустить production сервер та встановить всі залежності (рисунок 4,). Після цього Web-сервіс буде доступний на вашому домені.

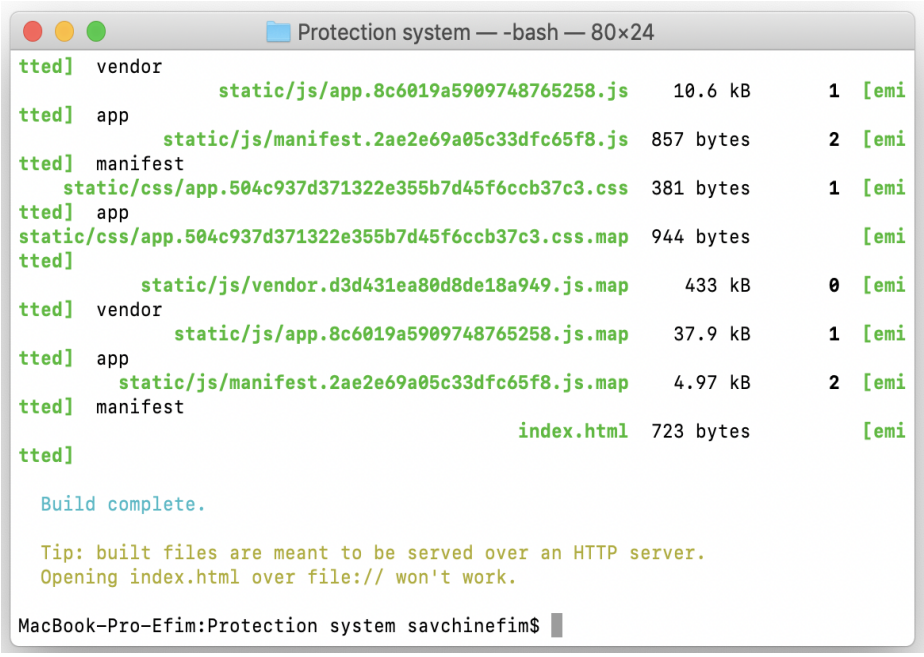

Рисунок 4.2 – Розгортка production версії додатку

4.2 Інструкція з використання програмного продукту

При вході на клієнтський додаток, користувачеві необхідно авторизуватися в системі щоб почати працювати в системі. На рисунку 4.3 зображена форма авторизації. Для авторизації необхідно ввести логін та пароль від існуючого аккаунту.

Дані авторизації буде відправлено на сервер Google Firebase, та, якщо дані коректні вам буде видано сесійний токен, з яким ви будете працювати у приватному кабінеті системи.

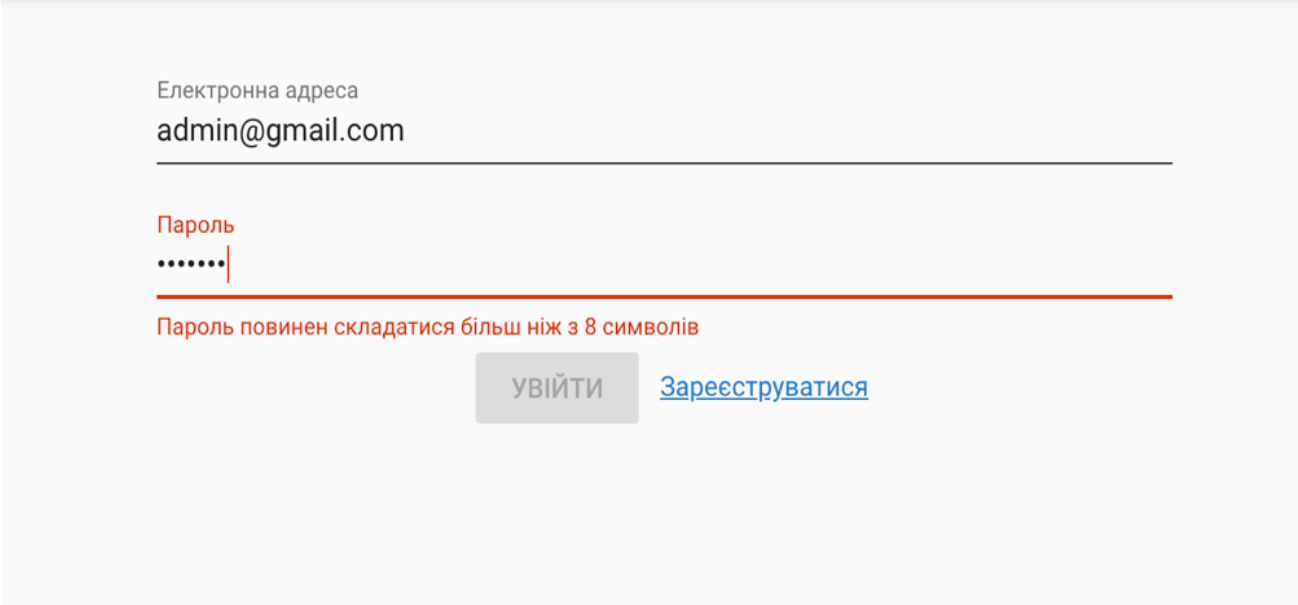

Рисунок 4.3 — Форма авторизації

У випадку якщо користувач ще ніколи не користувався системою або хоче створити новий профіль, то йому необхідно зареєструватися в системі. На рисунку 4.4 зображена форма реєстрації. При заповнені всіх полів форми стане доступна кнопка «зареєструватися», при натисненні на яку користувача буде додано до бази даних і йому буде видано унікальні ключі для захисту персональних даних, які не будуть доступні нікому, окрім лікуючого лікаря по вашому аккаунту і безпосередньо користувачу системи.

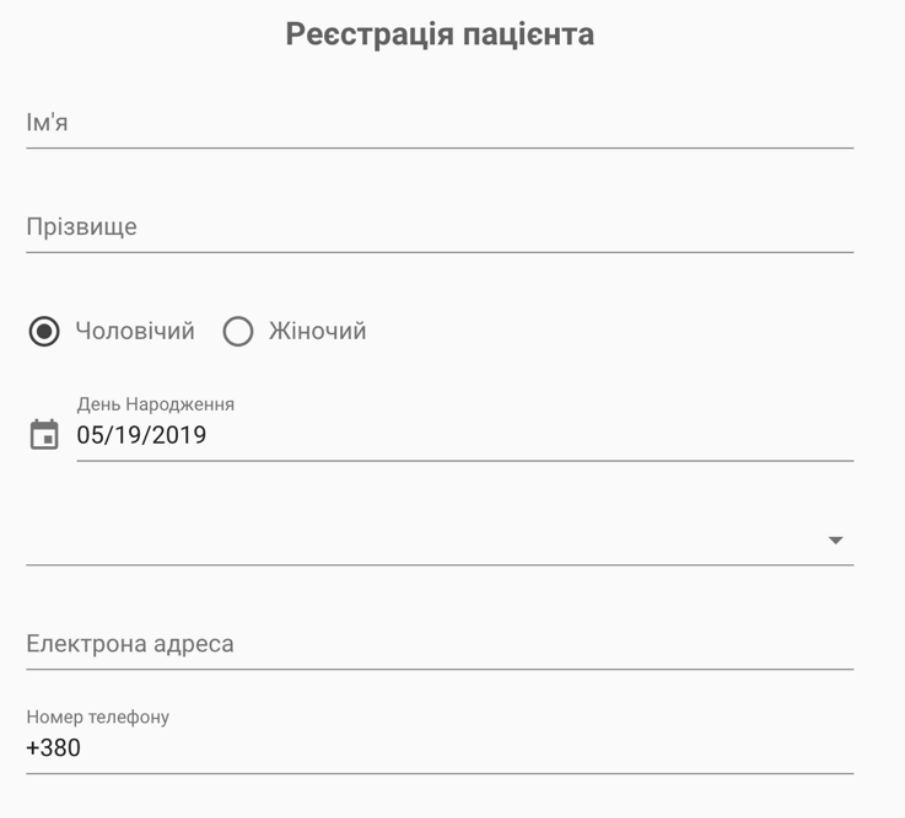

Рисунок 4.4 — Форма реєстрації пацієнта

Після того, як користувач авторизувався в системі, він отримує доступ до головного меню системи. За допомогою цього меню користувач має доступ до усіх сторінок системи, або має можливість вийти з свого профілю. Також він може перейти до свого основного робочого простору – кабінету користувача. На рисунку 4.5 зображене головне меню системи.

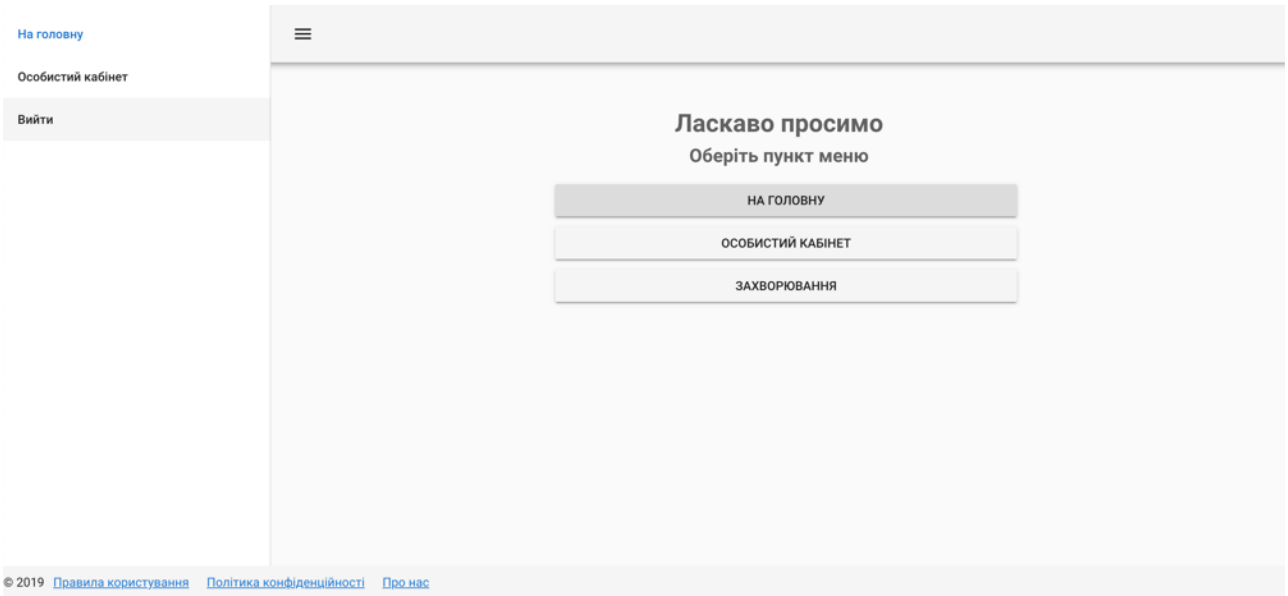

Рисунок 4.5 — Головне меню системи

Першим пунктом меню є особистий кабінет користувача, де відображено повну інформацію, та є можливість її редагування (рисунок 4.6)

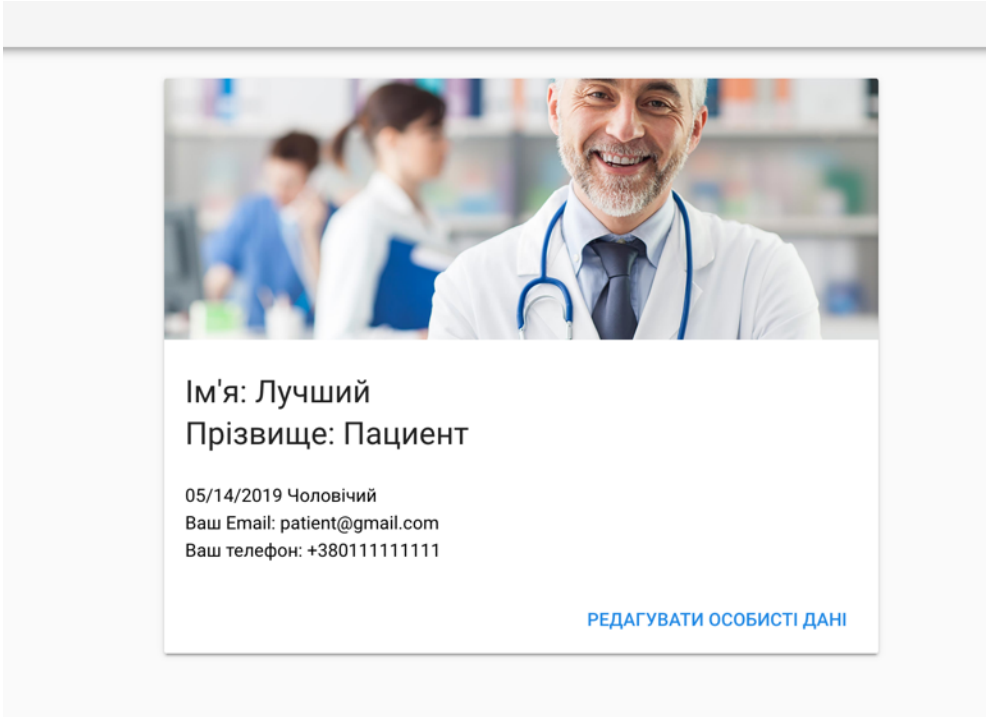

Рисунок 4.6 — Особистий кабінет користувача

У розділі «Захворювання» пацієнт може переглянути свої захворювання, діагнози та їх лікування (рисунок 4.7)

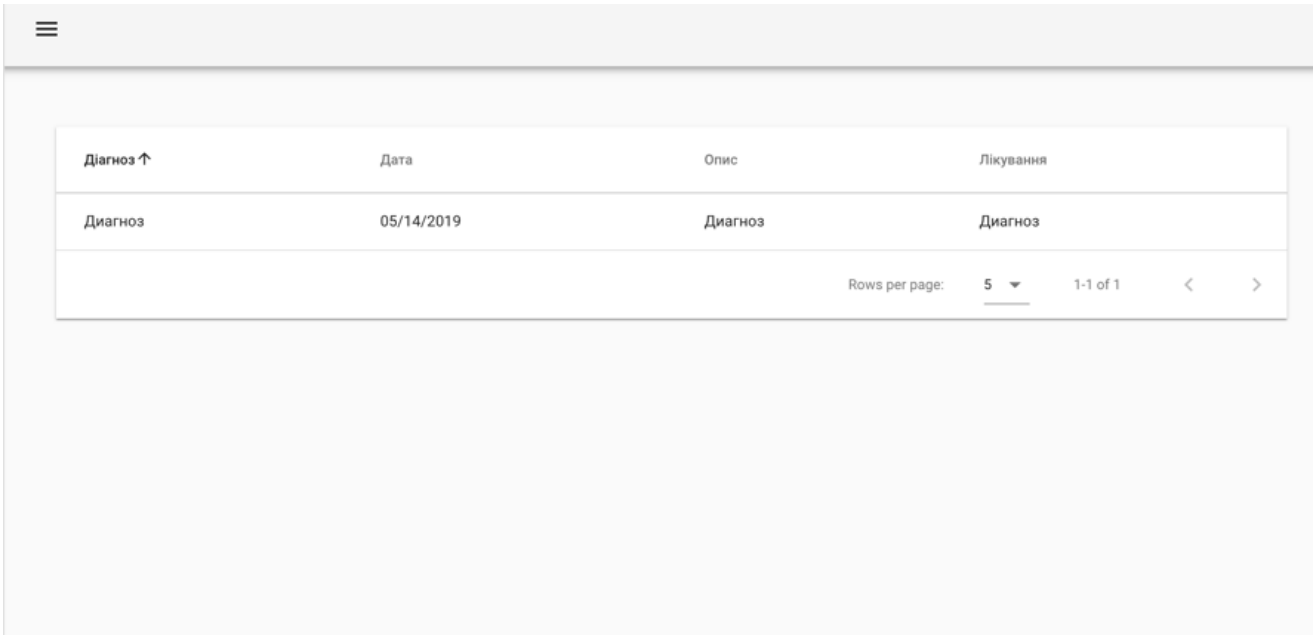

Рисунок 4.7 — Розділ «Захворювання»

Кабінет доктора також має приватний кабінет с можливістю редагування даних, аде головною функцією лікаря є можливість перегляду пацієнтів, закріплених за лікарем а також можливість редагувати історію захворювань. Приклад роботи цього модуля показано на рисунку 4.8.

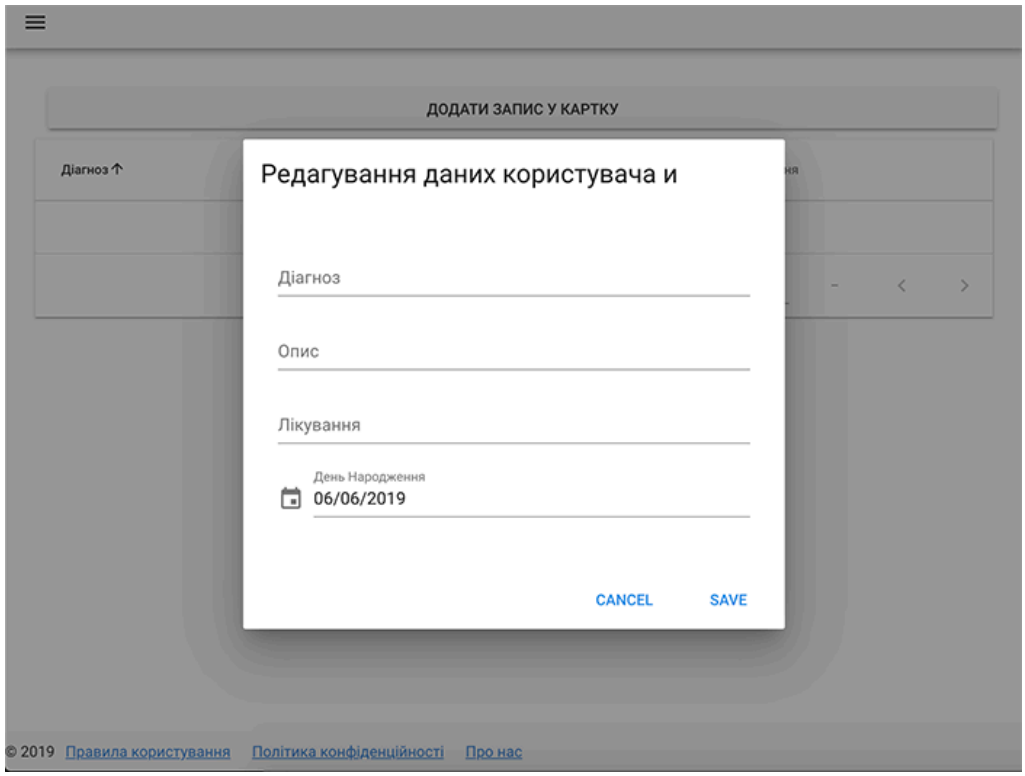

Рисунок 4.8 – Редагування історії захворювань пацієнта

При вході в адміністративний аккаунт меню систему буде виглядати так, як зображено на рисунку 4.9, а до стандартних функцій звичайного користувача будуть додані функції керування співробітниками та пацієнтами лікарні.

Рисунок 4.9 – Адміністративне меню додатку

Приклад адміністрування штату співробітників наведено на рисунку 4.10. На сторінці редагування можливо додати лікаря (тільки адміністратор може це зробити), редагувати дані лікаря, а також видалити користувача зі статусом лікаря з системи.

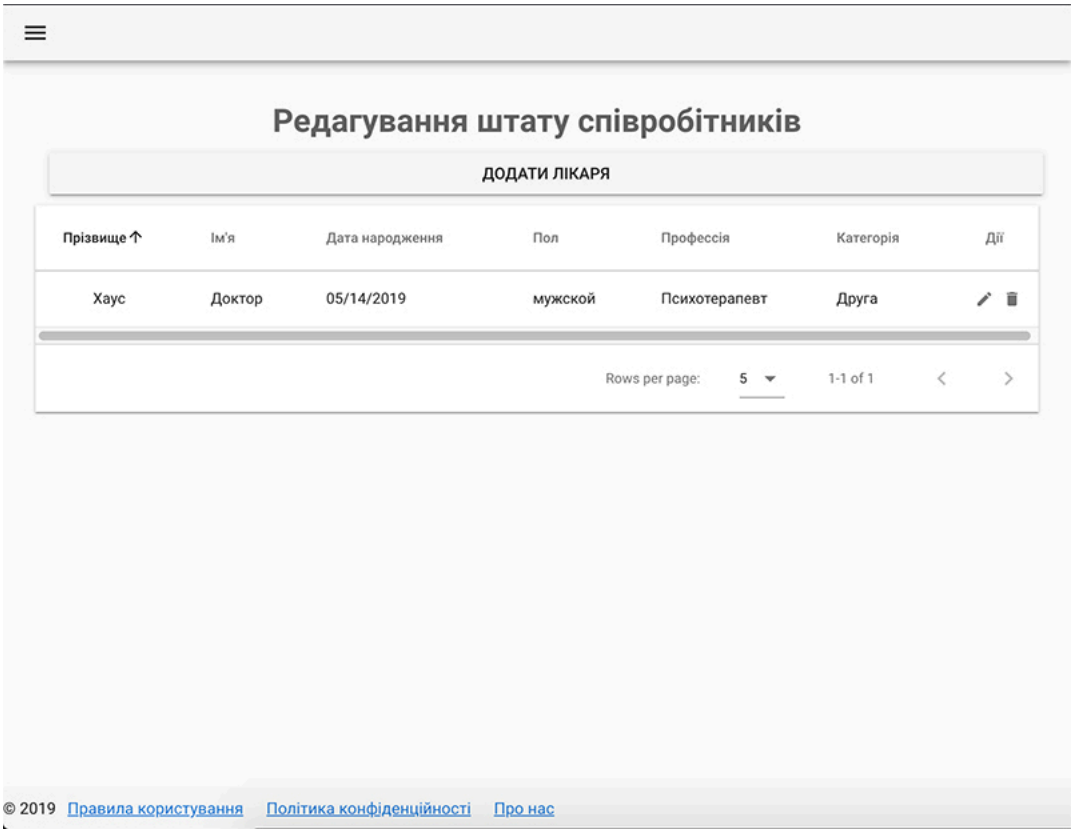

Рисунок 4.10 – Адміністрування штату співробітників

ВИСНОВКИ

В результаті проектування та написання дипломної роботи досліджено сучасні методики та принципи шифрування даних, проведено аналіз алгоритмів шифрування–дешифрування текстових даних.

Запропоновано розробити web-систему, захисту даних на базі алгоритму Ель–Гамаля, яка дозволить реєструвати користувачів, буде генерувати унікальні ключі, необхідні для застосування криптографічних методів, та буде зберігати всі дані в базі даних у зашифрованому вигляді. Для зберігання ключів запропоновано використовувати окремий сервер та таблицю бази даних, що гарантує більшу безпеку системи. Для реалізації обрано стандартний сучасний набір web-технологій: JavaScript, CSS, Vue.js, Nuxt, Node.js, MongoDB.

В ході виконання роботи було розроблено систему, яка представляє собою «розумну лікарню», та дозволяє безпечно слідкувати за своєю електронною карткою історії захворювань. Розроблений сервіс працює на всіх сучасних платформах, та має низькі, по сучасним міркам, технічні вимоги для завантаження та розгортування на стороні клієнта.

По темі дипломної роботи було написано тези на двох конференціях:

- сучасні проблеми сталого розвитку енергетики (тема доповіді – «Система захисту даних на базі алгоритму Ель–Гамаля»);
- "MATLAB та комп'ютерні обчисленняв освіті, науці та інженерії" (тема доповіді – «Застосування алгоритму Ель–Гамаля для захисту даних»)

Окрім цього отримано свідоцтво про реєстрацію авторського права на комп’ютерну програму «Information security system» №88390 від 10.05.2019, копію свідоцтва наведено в додатках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Саломаа А. Криптография с открытым ключом. Пер. с англ. – М.: Мир, 1995. – 318 с.
2. Смарт Н. Криптография. Москва: Техносфера, 2005. – 528 с.
3. Нильс Фергюсон, Брюс Шнайер Практическая криптография. Вильямс, 2005 г., 424 стр.
4. Конеев И. Р., Беляев А. В. Информационная безопасность предприятия.- СПб.: БХВ-Петербург, 2003.- 752с.
5. Мелюк А. А., Пазизин С. В., Погожин Н. С. Введение в защиту информации в автоматизированных системах. -М.: Горячая линия - Телеком, 2001.- 48с.
6. Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org/. Дата доступу: травень 2018. Назва з екрану.
7. Peter, H. Gregory Blocking Spam For Business For Dummies® (For Dummies (Computers)) / Peter H. Gregory. - Москва: ИЛ, 2016. - 636 с.
8. Бабаш, А. В. История криптографии. Часть I / А.В. Бабаш, Г.П. Шанкин. - М.: Гелиос АРВ, 2016. - 240 с.
9. Бабенко, Л. К. Современные алгоритмы блочного шифрования и методы их анализа / Л.К. Бабенко, Е.А. Ищукова. - М.: Гелиос АРВ, 2015. - 376 с.
10. Бабенко, Л.К. Современные интеллектуальные пластиковые карты / Л.К. Бабенко. - М.: Гелиос АРВ, 2015. - 921 с.
11. Болотов, А. А. Элементарное введение в эллиптическую криптографию. Протоколы криптографии на эллиптических кривых / А.А. Болотов, С.Б. Гашков, А.Б. Фролов. - М.: КомКнига, 2012. - 306 с.
12. Бузов, Геннадий Алексеевич Защита информации ограниченного доступа от утечки по техническим каналам / Бузов Геннадий Алексеевич. - М.: Горячая линия - Телеком, 2016. - 186 с.
13. Гамма Э. Приемы объектно-ориентированного проектирования; Книга по Требованию - Москва, 2002. - 376 с.

14. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования; Питер - Москва, 2013. - 368 с.
15. Гамма, Э.; Хелм, Р.; Джонсон, Р. и др. Приемы объектно-ориентированного проектирования; СПб: Питер - Москва, 2008. - 366 с.
16. Гамма, Э.; Хелм, Р.; Джонсон, Р. и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования; СПб: Питер - Москва, 2001. - 368 с.
17. Фримен Эрик , Фримен Элизабет , Сьерра Кэтти , Бейтс Берт Паттерны проектирования; Питер - Москва, 2013. - 322 с.
18. Software Requirements, Third Edition [Електронний ресурс]. – Режим доступу: https://www.processimpact.com/karls_books/SR3E/Software%20Requirements%20Third%20Edition%20Sample%20Chapters.pdf. Дата доступу: травень 2018. Назва з екрану.
19. Гаевский, А.Ю. 100% самоучитель. Создание Web-страниц и Web-сайтов. HTML и JavaScript / А.Ю. Гаевский, В.А. Романовский. - М.: Наука, 2015. - 464 с.
20. Дронов, В. JavaScript в Web-дизайне / В. Дронов. - М.: СПб: БХВ, 2014. - 880 с.
21. Кингсли-Хью, К.Э. JavaScript 1.5: учебный курс / К.Э. Кингсли-Хью. - М.: СПб: Питер, 2013. - 272 с.
22. Негрино JavaScript для начинающих / Негрино, Том. - М.: Огни, 2013. - 544 с.
23. Федоров, А.Г. JavaScript для всех / А.Г. Федоров. - М.: Машиностроение, 2012. - 384 с.
24. Anthony Gore — Full-Stack Vue.js 2 and Laravel 5 [Електронний ресурс]. — 2015. – Режим доступу: https://bit.ly/2OEODzR.
25. React – A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://reactjs.org/
26. Изучаем Node.js. – М.: Питер, 2013. – 400 с.
27. Кантелон, М. Node.js в действии / М. Кантелон. – М.: Питер, 2015. – 810 с.

28. Сухов, Кирилл Node.js. Путеводитель по технологии / Кирилл Сухов. – М.: ДМК Пресс, 2015. – 416 с.
29. Хэррон, Дэвид Node.js Разработка серверных веб–приложений на JavaScript / Дэвид Хэррон. – М.: ДМК Пресс, 2014. – 144 с.
30. Прохоренок, Николай HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Николай Прохоренок. - М.: БХВ-Петербург, 2012. - 912 с.

ДОДАТОК А

Система захисту даних на базі алгоритму Ель-Гамаля

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б 12-1	pages/index.vue static/encrypt.js static/decrypt.js	Основні компоненти
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б 13-1	Додаток В.doc	Опис програмного модуля

ДОДАТОК Б

Система захисту даних на базі алгоритму Ель-Гамаля

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б 12-1

Аркушів 8

Київ – 2019

```
//cryptoConstants.js

import bigInt from 'big-integer'

export const groups = [
  {
    id: 0,
    bits: 2048,
    g: 5,
    p:
bigInt('31737565950851398909263463161284348682160315370485844965773040301422571251968505
94171869047093884558132222357030125078884775784671141419664012211599207989977592557646698
34049791069892049442459039594755032328144507551012292392721234041299962841174554138915 94
98407431897698498100051102110917008633747018744296417816522047524120449714751944604661 04
87228872137880246356114822298913929231065471046437552758765145569157480161279053504872 05
59147146408276929935982236504198974200058608717718254958522101278495251103087932695351 01
75445491317225593650924247152361865371610166905907235359739289487545536749774030288543 37
787928323')
  },
  {
    id: 1,
    bits: 1024,
    g: 5,
    p:
bigInt('15981998168186168522874585527865336134217720450048582584829867222911531162978447
15979865069275999051892625858230826071783680592944382820206439021763948401397842604927 92
11424922151747987070054990955467918627397078408451918534136965214032617146686475638560 63
12523307581517823972926489963055849521932211018272883')
  },
  {
    id: 2,
    bits: 512,
    g: 5,
    p:
bigInt('13110251033247263553178100544755907143695412337941230657908852613728642884000122
74528240777836177539551726793813333650568633013492542937789714379092506 1003')
  },
  {
    id: 3,
    bits: 256,
    g: 5,
    p:
bigInt('81468211328722123306858066832132128014034308233325202328321941526153900552303')
  },
  {id: 4, bits: 128, g: bigInt('5'), p:
bigInt('187696290347723666271064587106794651083')},
  {id: 5, bits: 64, g: bigInt('5'), p: bigInt('11792624869596505283')},
]

export const selectedGroup = 5

export function getBits() {
  return groups[selectedGroup].bits
}

export function getP() {
  return groups[selectedGroup].p
}

export function getG() {
  return groups[selectedGroup].g
}

export function getX(p) {
  return bigInt.randBetween(p.divide(10), p);
}

export function getY(p, g, x) {
  return bigInt(g).modPow(x, p);
}
```

```
export function getK(p) {
  let k = bigInt.randBetween(p.divide(10), p);
  return k
}

export function getAlfa(k, g, p) {
  if (k) {
    return g.modPow(k, p);
  }
  return 'Создайте временный ключ';
}

//decrypt.js

import bigInt from "big-integer";

export function messageDecrypted(messageToDecrypt, alfa, x, p) {
  if (messageToDecrypt !== '' && alfa !== '' && x) {
    try {
      let beta = bigInt(messageToDecrypt.toString().trim());
      return bigInt(alfa.toString().trim())
        .modPow(p.minus(1).minus(x), p)
        .multiply(beta).mod(p);
    } catch (error) {
      // nothing
    }
  }
  return '-';
}

export function messageDecoded(messageDecrypted) {
  if (messageDecrypted !== '-') {
    return hexDecode(messageDecrypted.toString(16));
  }
  return 'Введите все необходимые данные';
}

export function hexDecode(str) {
  let j;
  let hexes = str.match(/.{1,2}/g) || [];
  let result = '';
  for (j = 0; j < hexes.length; j++) {
    result += String.fromCharCode(parseInt(hexes[j], 16));
  }
  return result;
}

export function decrypt(encryptedMessage, alfa, x, p) {
  encryptedMessage = encryptedMessage.split(',')
  let decoded = ''
  for(let i = 0; i < encryptedMessage.length; i++) {
    let decrypted = messageDecrypted(bigInt(encryptedMessage[i]), alfa, x, p)
    decoded += messageDecoded(decrypted)
  }
  return decoded
}
import { getAlfa, getG, getK, getP, getX, getY } from "./cryptoConstants";
import { decrypt } from './decrypt'
import bigInt from 'big-integer'

export async function crypto(
  message = 'Существуют две основные трактовки понятия «текст»',
  mode,
  parX,
  parY,
  parK,
  parAlfa
){
  let p = getP()
  let g = getG()
  let x = parX || getX(p)
```

```
  let y = parY || getY(p, g, x)
  let k = parK || getK(p)
  let alfa = parAlfa || getAlfa(k, g, p)

  if(mode === 1) {
    message = trans(message);
    return {
      message: encrypt(message, y, k, p),
      y,
      k,
      alfa,
      x
    }
  } else if(mode === 2) {
    return trans(decrypt(message, alfa, x, p), true)
  }
}

export function encrypt(text, y, k, p) {
  let encrypted = []
  let encStr = ''
  for(let i = 0; i < text.length; i++) {
    let mess = messageEncoded(text[i])
    encStr += (messageEncrypted(mess, y, k, p).value)
    if(i + 1 !== text.length) {
      encStr+= ','
    }
  }
  // let encryptedMessage = encStr.split(',')
  return encStr
}

//encrypt.js

export function messageEncrypted(messageEncoded, y, k, p) {
  if (messageEncoded !== '-' && y.toString().trim() !== '' && k) {
    try {
      return bigInt(y.toString().trim()).modPow(k, p)
        .multiply(messageEncoded).mod(p);
    } catch (error) {
      // nothing
    }
  }
  return 'Заполните все данные';
}

export function messageEncoded(message) {
  if (message.trim() !== '') {
    return bigInt(hexEncode(message.trim()), 16);
  }
  return '';
}

export function hexEncode(str) {
  let hex, i;
  let result = '';
  for (i = 0; i < str.length; i++) {
    hex = str.charCodeAt(i).toString(16);
    result += ('0' + hex).slice(-2);
  }
  return result;
}

let rus = "  щ   ш  ч  ц  ю  я  ё  ж  ъ  ы  э  а б в г д е з и й к л м н о п р с т у ф х
ь".split(/ +/g),
  eng = "  shh sh ch cz yu ya yo zh `` y' e` a b v g d e z i j k l m n o p r s t u f x
`".split(/ +/g)

function trans(text, engToRus) {
  var x;
  for(x = 0; x < rus.length; x++) {
```

```
    text = text.split(engToRus ? eng[x] : rus[x]).join(engToRus ? rus[x] : eng[x]);
    text = text.split(engToRus ? eng[x].toUpperCase() :
rus[x].toUpperCase()).join(engToRus ? rus[x].toUpperCase() : eng[x].toUpperCase());
  }
  if(engToRus) {
    return text
  }
  return text.match(/.{1,7}/g)
}
import firebase from 'firebase';

export default async function({ store, route, redirect }) {
  store.dispatch('loading/setLoading', true)
  await firebase.auth().onAuthStateChanged(async user => {
    if (user) {
      await store.dispatch('user/autoLogin', user)
      store.dispatch('loading/setLoading', false)
    }
    user !== null && route.name === 'login' ? redirect('/') : ''
    user !== null && route.name === 'register' ? redirect('/') : ''
    user === null && route.name !== 'login' ? redirect('/login') : ''
    user === null && route.name !== 'register' ? redirect('/login') : ''
  })
}

//index.vue

<template>
  <v-container bg fill-height grid-list-md text-xs-center class="flex">
    <v-layout row wrap align-end class="center">
      <v-flex
        xs12
        md6
        offset-md3
      >
        <h1>Ласкаво просимо</h1>
        <h2>Оберіть пункт меню</h2>
        <v-btn v-for="item in menuItems" :key="item.title" class="w-100" :to="item.to
=== '/profile' ? '/profile/' + userInfo._id : item.to" color="normal">{{item.title}}</v-
btn>
        <v-btn v-if="userInfo.status === '1'" class="w-100" :to="'diseases/' +
userInfo._id" color="normal">Захворювання</v-btn>
      </v-flex>
    </v-layout>
  </v-container>
</template>

<script>
export default {
  name: 'index',
  computed: {
    loading() {
      return this.$store.state.loading.loading
    },
    menuItems() {
      return this.$store.state.user.menuItems
    },
    userInfo() {
      return this.$store.state.user.userInfo
    }
  }
}
</script>

<style>
h1, h2, h3, h4, p {
  color: #616161;
}
h2{
  padding-bottom: 10px;
}
```

```
.w-100 {
  width: 100%;
}
</style>

//model.crypto.js
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
let Post = new Schema({
  id: {
    type: String
  },
  x: {
    type: String
  },
  y: {
    type: String
  },
  k: {
    type: String
  },
  alfa: {
    type: String
  }
},{
  collection: 'hospital-crypto'
});

module.exports = mongoose.model('Crypto', Post);

//model.patient.js

const mongoose = require('mongoose');
const Schema = mongoose.Schema;
// Define collection and schema for Post
let Post = new Schema({
  status: {type: String},
  firstName: {type: String},
  secondName: {type: String},
  gender: {type: String},
  birthDate: {type: String},
  email: {type: String},
  phone: {type: String},
  profession: {type: String},
  category: {type: String},
  password: {type: String}
},{
  collection: 'hospital-doctors'
});

module.exports = mongoose.model('Doctor', Post);

//api.patient.js
const express = require('express');
const postRoutes = express.Router();
let Post = require('./model.patient');
postRoutes.route('/add').post(function (req, res) {
  let post = new Post(req.body);
  post.save()
    .then((response) => {
      res.status(200).json({'business': response._id});
    })
    .catch(() => {
      res.status(400).send("unable to save to database");
    });
});
postRoutes.route('/').get(function (req, res) {
    Post.find(function(err, posts){
    if(err){
      res.json(err);
    }
```

```
    else {
      res.json(posts);
    }
  });
});
postRoutes.route('/getUsersByDocId/:id').get(function (req, res) {
  Post.find(function(err, posts){
    if(err){
      res.json(err);
    }
    else {
      let postList = []
      for (let key in posts) {
        if (posts[key].doctor === req.params.id) {
          postList.push(posts[key])
        }
      }
      res.json(postList);
    }
  });
});
postRoutes.route('/findUser/:id').get(function (req, res) {
  let id = req.params.id;
  Post.find({email: id}, function (err, post){
      if(err) {
        res.json(err);
      }
      res.json(post);
  });
});
postRoutes.route('/findUserById/:id').get(function (req, res) {
  let id = req.params.id;
  Post.findById(id, function (err, post){
    if(err) {
      res.json(err);
    }
    res.json(post);
  });
});
postRoutes.route('/update/:id').post(function (req, res) {
    Post.findById(req.params.id, function(err, post) {
    if (!post)
      res.status(404).send("data is not found");
    else {
      post.firstName = req.body.firstName
      post.secondName = req.body.secondName
      post.gender = req.body.gender
      post.birthDate = req.body.birthDate
      post.email = req.body.email
      post.phone = req.body.phone
      post.password = req.body.password

      post.save().then(() => {
        res.json('Update complete');
      })
      .catch(() => {
            res.status(400).send("unable to update the database");
      });
    }
  });
});

postRoutes.route('/delete/:id').delete(function (req, res) {
    Post.findByIdAndRemove({_id: req.params.id}, function(err){
        if(err) res.json(err);
        else res.json('Successfully removed');
    });
});

module.exports = postRoutes;
```

ДОДАТОК В

Система захисту даних на базі алгоритму Ель-Гамаля

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б 13-1

Аркушів 8

Київ - 2019

АНОТАЦІЯ

Додаток представляє собою опис частини, яка забезпечує роботу з БД, адже БД – це структурна одиниця програми, що зв’язує вся модулі програми. Призначенням БД є зберігання зашифрованої інформації та ключів користувачів, які генерується одразу при реєстрації. Модуль надає можливість отримувати необхідні дані, оновлювати та видаляти необхідні записи. Модуль написано мовою програмування Node.js, з використанням технології Express.js та системи не реляційних баз даних MongoDB.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається один з програмних модулів системи — модуль для роботи з БД з кодом, написаним у додатку УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51160_19Б 12-1, що міститься у директорії «api», яка знаходиться у корені проекту. Модуль реалізовано за допомогою бібліотеки Mongoose для MongoDB. Модуль призначений для управління підсистемами, які відповідають за шифрування-дешифрування даних, та їх запису у БД, та їх зчитування. Користувач має можливість зареєструватися в системі, редагувати, додавати та редагувати дані.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням модулю для роботи з БД є збереження інформації для клієнтської частини та безпосереднього прийняття даних введення та обмін інформацією із клієнтським інтерфейсом. Використання такого шаблону дозволяє створювати програмне забезпечення, де інтерфейс і логіка роботи модуля для БД є незалежними компонентами, що дає можливість використовувати його для зменшення навантаження на клієнтську частину системи.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Слідкувати за зміною інформації про підсистеми є головним завданням модуля. Переході від сторінки до сторінки модуль повертає всю інформацію, яка міститься в БД ссаме для цього маршруту, для відображення даних на користувацькому інтерфейсі. Також модуль оброблює інформацію, надаючи змогу додавати запис, змінювати або видаляти необхідний.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у середовищі розробки Microsoft Visual Studio 2017, що забезпечує набір сервісних функцій та графічний діалог з користувачем, на комп’ютері, що використовував операційну систему Windows 10. З БД, комп’ютер з’єднувався за допомогою пакету SqlClient.

В якості СКБД було використано MS SQL Server.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програмний модуль реалізований як окремий додаток, який забезпечує існування клієнтської частини та бізнес-логіки окремо одне від одного. Запуск додатків проводиться окремо, що дозволяє розмістити додатки на різних машинах.

Для використання даного модулю потрібно розмістити його на хостингу та надати клієнстському застосунку права на використання модулю.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для модуля є інформація, яку користувач вводить в додатку (реєстраційні дані, логін і пароль, захворювання, інформація про користувача, тощо).

Вихідними даними програмного модуля є необхідні записи, які потрібні для користувача.

ДОДАТОК Г

Система захисту даних на базі алгоритму Ель-Гамаля

Копія авторського свідотства

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51160_19Б

Аркушів 8

Київ – 2019